ENCICLOPEDIA DEL BASIC

LeG

SPECTRUM 3

Cómo simplificar y mejorar los programas



ceac

SPECTRUM 3

Cómo simplificar y mejorar los programas

ENCICLOPEDIA
DEL
BASIC

SPECTRUM 3

Cómo simplificar y mejorar los programas



Perú, 164 - 08020 Barcelona - España

Primera edición: Enero 1987

ISBN 84-329-7713-6 (Rústica)
ISBN 84-329-7718-7 (Rústica especial)
ISBN 84-329-7723-3 (Cartoné)
ISBN 84-329-7730-6 (Cartoné especial)
ISBN 84-329-7726-8 (Cartoné, Obra completa)
ISBN 84-329-7727-6 (Cartoné, Obra completa especial)

Depósito Legal: B-44039 - 1986

Impreso por
GERSA, Industria Gráfica
Tambor del Bruc, 6
08970 Sant Joan Despí (Barcelona)

Printed in Spain
Impreso en España

Contenido

Aclaración importante

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema,* lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Cómo estudiar en esta Enciclopedia

Al comenzar cada Capítulo encontrará un *esquema de contenido*. La finalidad de este esquema es doble:

— Por una parte, le ofrece una visión panorámica de todos los temas que va a estudiar en ese capítulo.

— Por otra, si posteriormente debe repasar algún punto determinado, le facilitará el poder localizarlo.

Leálos despacio para tener esta visión panorámica.

Después del esquema de contenido, cada capítulo comienza definiendo sus objetivos. De esta manera usted sabrá desde el principio lo que aprenderá en él. También le servirá como referencia para saber si el objetivo marcado ha sido alcanzado por usted.

A lo largo de los capítulos y al final de los mismos encontrará unos *resúmenes*, seguidos de unos *ejercicios de autocomprobación*. Su finalidad es hacer un alto en el camino y recapitular lo estudiado desde la última parada. Al mismo tiempo, los ejercicios de autocomprobación le servirán para que usted mismo compruebe si ha asimilado los conceptos estudiados y si está en disposición de continuar adelante o, por el contrario, si es necesario volver a repasar algo antes de seguir. La solución a los ejercicios de autocomprobación la encontrará al final de la primera parte del volumen.

Le recomendamos también que, cuando deba interrumpir su estudio, lo haga siempre al final de un capítulo o al terminar de resolver alguno de los grupos de ejercicios de autocomprobación que aparecen a lo largo de las lecciones.

Al hablar de la composición de la Enciclopedia, le dijimos que cada volumen se componía de dos partes. La del texto principal o estudio del BASIC estándar, y la de «Práctica con el microordenador», que viene a ser un complemento de la anterior. Ahora que va a comenzar a estudiarlo comprenderá enseguida la razón de esta comparación.

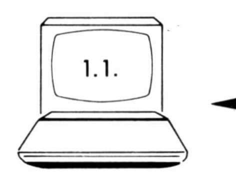
Cuando se aprende un idioma es frecuente que uno se encuentre con la sorpresa de que en determinadas regiones aquello que uno aprendió a decirlo de una manera se diga de otra. Con el lenguaje de programación BASIC pasa lo mismo. Cada fabricante introduce en el uso de su ordenador un dialecto distinto, que normalmente coincidirá en su mayor parte con el

BASIC estándar, pero tendrá suficientes características especiales como para que el que comienza se encuentre ante su ordenador sin saber qué hacer en determinados momentos.

Por eso, como nuestra intención es que aprenda BASIC y que lo practique con *su microordenador* hemos utilizado la Enciclopedia de modo que usted no se encuentre perdido en ningún momento. Así, al estudiar la lección del BASIC estándar es como si dijéramos: *Esto se dice así normalmente en lenguaje BASIC.* Sin embargo, cuando haya diferencias, añadiremos en el capítulo «Prácticas con el microordenador»: *Ojo, que esto mismo en su microordenador se dice de esta forma.* De este modo, podrá dialogar tranquilamente con su microordenador sin desagradables sorpresas y traducir al lenguaje que entiende su microordenador un programa que en BASIC estándar pueda estar escrito de forma un poco diferente.

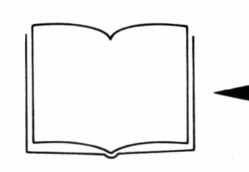
Concretamente, en el estudio de las dos partes debe proceder del modo siguiente:

— Comience siempre cada capítulo por el texto principal y continúe hasta que encuentre esta viñeta en el margen izquierdo:



Verá que en la pantalla de la viñeta aparecen unos números. Concretamente 1.1. Esto indica que debe dejar en este momento el capítulo que está estudiando, y pasar al apartado 1.1. de «Prácticas con el microordenador».

— Continúe ahora el estudio de «Prácticas con el microordenador» hasta que encuentre esta otra viñeta:



En ella se le remite de nuevo al capítulo que dejó anteriormente, para continuar donde lo interrumpió. También aquí se le indica el sitio exacto donde debe continuar. En todo caso, le recomendamos que deje siempre una señal en la página donde interrumpe el estudio. Así tendrá siempre la página localizada.

Observará el lector de esta «Enciclopedia del BASIC» que la numeración de capítulos es correlativa, iniciándose en el Tomo I de la misma y finalizando en el Tomo V y último.

Hemos creído conveniente tal continuidad, en primer lugar por el carácter secuencial de la Enciclopedia, al ser los temas que la constituyen correlativos entre sí y formando un todo inseparable; y en segundo lugar porque estimamos que ello facilita la *rápida localización de cualquier tema*, lo que, al tratarse de una obra de consulta frecuente, supone una indudable ventaja para el lector.

Parte I
BASIC

Capítulo 9
• La estructura de repetición y el estudio de los bucles

ESQUEMA DE CONTENIDO

Objetivos	
La estructura de repetición	
La instrucción FOR y la instrucción NEXT	Instrucción FOR Instrucción NEXT Funcionamiento del bucle Diversos casos de bucles FOR
Bucle escalonado	
Bucles decrecientes	
Bucles anidados	
Bucles de espera	
El GOTO y los bucles	Salida de los bucles Entrada de los bucles
Posibilidades de los bucles FOR/NEXT	Propiedades generales Utilización peligrosa del bucle

9.0 OBJETIVOS

Iniciamos con este capítulo una segunda etapa en el estudio de la programación. Consiste en el estudio de las instrucciones que dan un cierto confort a la programación.

Estas instrucciones no son en absoluto necesarias para la programación. Con todas las instrucciones que conoce en este momento puede construir cualquier programa que comporte sólo cálculo, excepto, claro está, cuestiones gráficas y de sonido.

La primera instrucción que vamos a desarrollar en este capítulo nos introduce en una instrucción de amplia aplicación en programación: el bucle.

La potencia de esta instrucción se debe a que permite especificar de una manera muy cómoda los procesos repetitivos que son muy frecuentes en los programas.

Hay muchos procesos que se repiten al menos conceptualmente. Por ejemplo, el hacer una factura consiste en calcular una línea detrás de la otra con el mismo esquema de cálculo pero con distintos artículos, nombres, precios y cantidades.

Diremos que el proceso es rutinario y repetitivo.

Este tipo de trabajos son muy frecuentes en la elaboración de programas en que la semejanza del proceso de cálculo es manifiesta, pero que deben realizarse con datos distintos.

El BASIC tiene dos instrucciones específicamente diseñadas para facilitar la programación de estos procesos: son las instrucciones FOR y NEXT.

Debe poner mucha atención, pues es el primer caso que se nos presenta de dos instrucciones que deben actuar coordinadamente, es decir, no tiene sentido utilizar una sin utilizar la otra.

Evidentemente todas las instrucciones del programa deben actuar de una forma coordinada para conseguir el fin propuesto, pero en el caso de las instrucciones FOR y NEXT, esta coordinación es más intensa, debido a que una no tiene sentido sin la otra; las demás instrucciones pueden actuar aisladamente.

Puede parecer extraño que se dedique un capítulo a una sola instrucción; pero es debido a que, dada su importancia, es necesario asimilar muy bien su forma de actuar.

Hemos desarrollado el tema de una manera muy pormenorizada, considerando casos cada vez más complicados para mostrar gradualmente la potencia de la instrucción.

Estúdielos con cariño aunque le parezcan demasiado sencillos. La experiencia nos indica que, a pesar de su aparente sencillez, se cometen muchos errores en programación debido a la utilización equivocada de la instrucción, que provienen de una mala utilización de los conceptos.

Sobre todo ponga atención en las cosas que debe evitar para que el funcionamiento de la instrucción sea correcto.

Queremos advertirle que hemos utilizado la comparación del mecanismo de funcionamiento de las instrucciones FOR y NEXT con el mecanismo propio de estas instrucciones, pero desarrollado con las instrucciones que conocemos en el momento de iniciar el capítulo.

Aprecie esta comparación en dos aspectos:

En primer lugar, le ayudarán a comprender de un modo exacto el concepto de bucle y su funcionamiento.

En segundo lugar, tenga en cuenta que las instrucciones FOR y NEXT siempre se pueden sustituir por el conjunto de estas otras instrucciones más elementales. Esta característica le puede ayudar a detectar los errores cuando un bucle no funcione.

9.1 LA ESTRUCTURA DE REPETICION

En capítulos anteriores hemos aprendido diversas instrucciones, con distintas finalidades: entrada de datos, cálculo, impresión de resultados, control... En este capítulo vamos a referirnos de nuevo a estas últimas, para conocer una de sus posibilidades más interesantes: la construcción de estructuras de repetición.

Aunque hemos visto ya algunos programas para construir bucles —bucle es otra manera de llamar a las estructuras de repetición—, vamos a dedicar todo este capítulo a este tipo de estructuras, profundizando en su utilización, y extendiéndonos en sus posibilidades, ya que estas tareas constituyen la base de aplicación de los ordenadores.

Para ejecutar un proceso más de una vez, conocemos la instrucción GOTO. Si esta instrucción se sitúa al final de un proceso cualquiera, de forma que nos remita el control de nuevo al inicio del mismo, obtendremos la ejecución repetida de este proceso. Veamos un caso cualquiera; por ejemplo, la escritura de un texto. Para imprimir un texto, como la frase «HOLA, QUE TAL?», escribiremos:

```
NEW
10   PRINT "HOLA, QUE TAL?"
```

y ejecutaremos el programa, con lo que obtendremos el texto

«HOLA, QUE TAL?»

escrito en la pantalla, tal como queríamos.

Si ahora deseamos que se repita el proceso, tal como hemos dicho, deberemos escribir:

```
20   GOTO 10
```

de forma que el programa nos queda:

```
10   PRINT "HOLA, QUE TAL?"
20   GOTO 10
```

La estructura de repetición también se llama bucle

Si ahora escribimos RUN, desencadenamos un bucle sin final, de forma que el texto se escribirá repetidamente en la pantalla hasta que detengamos el programa, pulsando la correspondiente tecla de interrupción.

Hemos generado una estructura de repetición, pero evidentemente una estructura descontrolada. Esto no nos resulta útil. Lo que nos interesaría es repetir el proceso pero no indefinidamente, sino bajo unas condiciones determinadas, que nosotros pudiéramos fijar. Así por ejemplo, nos interesará repetirlo un número de veces determinado: 10, 20, 1000, ..., o bien repetirlo únicamente si se cumplen unas condiciones especiales. Lo que importa, pues, es tener el control: poder indicar cuándo se ha de realizar y cuántas veces se ha de repetir un proceso.

Para «decidir» en un programa, conocemos otra instrucción, que nos permite elegir entre una o varias alternativas. Esta instrucción, que también hemos visto en anteriores unidades didácticas, es la instrucción IF...THEN...

Dado que esta instrucción nos permite escoger ante una disyuntiva, la podemos utilizar para controlar el bucle. La forma de hacerlo es dar la indicación de seguir el bucle únicamente si se cumplen las condiciones que nosotros hayamos establecido.

La variable de control del bucle

Hemos dicho que una posible condición es que se hayan dado un número de vueltas determinado, lo que equivale a haber repetido el proceso este número de veces. Esto implica que se necesita de alguna forma controlar las vueltas que se han dado. Para ello utilizaremos una variable que, dada su finalidad, se denominará variable de control. Veamos cómo podemos controlar el bucle que teníamos escrito:

```
10 PRINT "HOLA, QUE TAL?"
20 GOTO 10
```

Supongamos que lo queremos controlar de forma que nos escriba el texto diez veces, y que se pare a continuación. En primer lugar necesitaremos incluir un «contador» en el bucle, una variable que se incrementa en una unidad cada vez que escribamos el texto, de forma que cuando esta variable alcance el valor 10 —lo habremos escrito diez veces— el control se transfiera fuera del bucle. La estructura básica del bucle ya la tenemos:

```
10 PRINT "HOLA, QUE TAL?"
20 GOTO 10
```

Debemos ahora incluir las instrucciones necesarias para realizar el control.

Veamos en primer lugar la cuestión del contador. Utilizamos la variable I como variable de control. Cuando empezamos el proceso le asignamos el valor 1, dado que vamos a efectuar la primera vuelta. Así, antes de entrar en el bucle haremos:

I = 1

A continuación escribiremos el texto una vez. Cuando se haya escrito, habrá que incrementar el contador. Esto equivaldría a hacer:

I = 2

A la siguiente vuelta, se escribirá de nuevo el texto. Ahora deberá hacerse:

I = 3

Y de esta forma se irá repitiendo el proceso. El contador deberá tomar sucesivamente los valores 1, 2, 3, 4, 5, ...

Es decir, el valor de la I se incrementa a cada vuelta en una unidad. Esto corresponde a sumar uno al valor de la I, es decir, hacer I+1 a cada vuelta. El valor resultante lo podemos almacenar de nuevo sobre la propia variable I. La instrucción para ello en BASIC será:

```
LET I = I+1
```

La expresión I=I+1 no tiene sentido desde el punto de vista matemático. Sin embargo, en BASIC conocemos ya su significado; el signo igual (=) no tiene el sentido matemático de igualdad, sino que tiene sentido de asignación. Es decir, se toma el valor de la variable I, se le suma la unidad, y el resultado se asigna de nuevo sobre la misma variable. Esta queda modificada; su valor actual será el que tenía más la unidad.

Inicialización de la variable de control

El valor de la variable I corresponderá, pues, al número de vueltas que haya dado el bucle. El programa quedará:

```
 5   LET I = 1
10   PRINT "HOLA, QUE TAL?"
15   LET I = I+1
20   GOTO 10
```

Si queremos además ver cómo va incrementándose el contador, añadiremos una instrucción para que nos lo escriba:

```
12   PRINT I
```

El resultado del programa será ahora:

HOLA, QUE TAL?
1
HOLA, QUE TAL?
2
HOLA, QUE TAL?
3
...

Ahora, una vez hemos establecido el sistema de contar vueltas, nos hace falta añadir la instrucción de control, de forma que podamos salir del bucle cuando se haya escrito diez veces el texto. Así, como una instrucción IF...THEN..., comprobaremos si ya hemos dado 10 vueltas, es decir, si

$$I > 10$$

y en caso afirmativo, saldremos del bucle.

Para el ejemplo que estamos viendo, el proceso consiste en realizar una escritura. Por tanto, después de escribir, se controla si debemos continuar o no. Si se cumplen las condiciones fijadas —que en este caso dependerán del valor de la variable de control, que es lo mismo que decir el número de repeticiones hechas— se devuelve el control al principio del bucle, donde se efectúa una nueva ejecución de la escritura.

En el ejemplo que estamos viendo, para la escritura del texto, la instrucción de control se deberá situar en la línea del GOTO, es decir, la línea 20. Escribiremos entonces:

```
20    IF I<= 10 THEN GOTO 10
```

El programa completo quedará ahora:

```
 5  LET I=1
10     PRINT "HOLA QUE TAL?"
12     PRINT I
15     LET I=I+1
20     IF I<= 10 THEN GOTO 10
```

Observe que la pregunta se ha colocado en forma negativa, es decir, es equivalente a NOT I>10, si es afirmativa se vuelve al inicio del bucle.

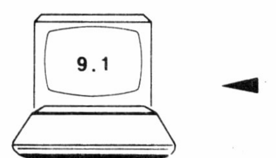
En resumen, los bucles o estructuras de repetición se construyen con los ingredientes siguientes:

1. Una variable de control que debe inicializarse.

2. Esta variable de control se incrementa en cada una de las vueltas, después de ejecutar la acción que quiere repetirse.

3. Una pregunta o decisión del tipo IF...THEN, que decide si el bucle debe continuarse o no.

4. Por fin, un cuerpo del bucle que consiste en la secuencia de instrucciones a realizar en cada vuelta. En el ejemplo que hemos puesto este cuerpo es corto, son las dos instrucciones PRINT, pero puede ser mucho más largo.



9.2 LA INSTRUCCION FOR Y LA INSTRUCCION NEXT

Dado que los procesos de repetición son sumamente frecuentes, el BASIC dispone de una manera más cómoda de escribirlos, utilizando las instrucciones FOR y NEXT.

La instrucción FOR sirve para indicar el principio del bucle y fijar las condiciones iniciales y finales de la variable de control del mismo.

La instrucción NEXT marca el final del cuerpo del bucle, indicando el alcance del mismo.

Entre ambas se escribe la acción o acciones a repetir. Así, de una forma general un bucle está constituido por tres partes:

Instrucción FOR → Inicio del bucle

Acción a repetir

Instrucción NEXT → Final del bucle

La acción a repetir y la instrucción NEXT suelen escribirse más hacia la derecha que la instrucción FOR. Esto se hace para indicar las instrucciones que forman parte del bucle, resaltando el alcance del mismo.

Es importante que se dé cuenta de que ambas instrucciones actúan de modo coordinado.

Actuación coordinada del FOR y el NEXT

Es decir, no es posible pensar en una instrucción FOR sin la NEXT y viceversa. De alguna manera actúan del mismo modo que los paréntesis. La instrucción FOR abre el paréntesis y la instrucción NEXT lo cierra. Un texto quedará incompleto si abrimos un paréntesis y no lo cerramos o cerramos un paréntesis sin haberlo abierto.

Debemos señalar que se trata de una comodidad, pues se pueden construir bucles con las instrucciones que se han visto hasta ahora; es decir, mediante la asignación, el GOTO y el IF...THEN. Por esta razón, todo lo que digamos se puede comparar con el programa que hemos visto en el apartado anterior, aunque para establecer una comparación más estricta, utilizaremos la siguiente forma del programa:

```
 5 LET I=1 : LET F=10
10   PRINT "HOLA QUE TAL?"
12   PRINT I
15   LET I=I+1
20   IF I<= F  THEN GOTO 10
```

La diferencia que hay con el programa anterior es que se ha introducido la variable F para indicar el final del contador, en lugar de hacerlo en la instrucción 20.

Vamos a ver con detalle la forma de escribir y de utilizar estas instrucciones.

9.2.1 Instrucción FOR

La forma general de escribir una instrucción es:

FOR variable = primer valor TO último valor

Por ejemplo, podemos escribir:

```
FOR I = 1 TO 10
```

cuyo significado es:

para I igual a 1 hasta 10

La función de esta instrucción es cuádruple:

Funciones del FOR

1. Indicarnos que se inicia un bucle (por lo tanto que abrimos una especie de paréntesis, que más tarde deberemos cerrar).

2. Definir la variable que se utilizará como control, en este caso la I (si la variable está definida no ocurre nada).

3. Inicializar el valor de la variable de control al valor inicial indicado en el FOR. En este caso se le da el valor 1.

4. Inicializar el valor que servirá de final del bucle. En este caso se le da el valor 10.

Si se compara con el programa que hemos realizado sin la instrucción FOR, ésta equivale a la instrucción 5, en donde definimos los valores de I y de F, como valores inicial y final respectivamente.

9.2.2 Instrucción NEXT

La forma general de escribir esta instrucción es:

NEXT variable

La variable debe ser la de control del bucle. La palabra NEXT significa *siguiente* en inglés.

La función de esta instrucción es triple:

Funciones del NEXT

1. Cierra el bucle iniciado anteriormente con la variable de control especificada en la instrucción NEXT (por lo tanto, cierra el paréntesis que hemos abierto anteriormente en una instrucción FOR).

2. Incrementa el valor de la variable de control.

3. Ejecuta la pregunta de *si el valor de la variable control sobrepasa al valor final.* En caso de que sea cierta, continúa con la instrucción siguiente. Si la afirmación es falsa, envía el control del programa a la instrucción siguiente a la instrucción FOR, que tiene como variable de control a la especificada en el NEXT.

En el programa que hemos tomado como ejemplo se corresponde con las instrucciones de las líneas 15 y 20.

Así, un bucle que deba efectuar 10 repeticiones, se escribirá:

```
FOR I = 1 TO 10
    Accion a realizar
    NEXT I
```

Justamente éste es el caso de nuestro ejemplo, el cuerpo de las acciones a realizar se sitúa entre las instrucciones FOR y NEXT, que corresponde a las instrucciones 10 y 12.

Estas instrucciones situadas entre el FOR y el NEXT se repetirán tantas veces como indique la variable de control.

La comodidad que representa la utilización del FOR y el NEXT en lugar de las instrucciones elementales, consiste precisamente en expresar de una manera compacta lo que se da con mucha frecuencia en programación. Además observe que en el caso del FOR y NEXT no es necesario utilizar un GOTO; éste conlleva la necesidad de recordar un número de línea mientras se escribe el programa.

9.2.3 Funcionamiento del bucle

El funcionamiento de un bucle, controlado mediante estas instrucciones, es el siguiente:

En primer lugar, la instrucción FOR indica las condiciones de ejecución del bucle, fijando los valores que debe tomar la variable de control del mismo.

Para realizar un seguimiento más cuidadoso de lo que ocurre a continuación damos el programa propuesto como ejemplo, escrito con la instrucción FOR y NEXT y sin ella:

```
 5 LET I=1 : LET F=10            5 FOR I = 1 TO 10
10   PRINT "HOLA QUE TAL?"      10   PRINT "HOLA QUE TAL?"
12   PRINT I                    12   PRINT I
15   LET I=I+1
20   IF I<= F THEN GOTO 10      20   NEXT I
```

Primera vuelta: Pasamos por la instrucción FOR, que asigna a la variable de control el valor inicial indicado en la misma. La equivalencia de

las dos instrucciones 5 en ambos programas nos muestra las operaciones exactas que realiza la instrucción FOR.

Se ejecutan las instrucciones del cuerpo del bucle hasta llegar a la instrucción NEXT. La correspondencia de los programas indica que se imprime el mensaje y el valor de la variable de control.

La instrucción NEXT incrementa en primer lugar el valor de la variable de control. En la comparación de programas es equivalente a la instrucción 15.

A continuación se compara este valor con el último que debe tomar. Pueden presentarse entonces dos casos:

— Si el valor asignado es superior al valor máximo, el bucle debe terminar. El control es automáticamente transferido fuera del mismo. Esto quiere decir que pasará a la línea situada a continuación de la instrucción NEXT que marca el final del bucle.

— Si el valor asignado es inferior o igual al valor máximo, nos encontramos en situación de ejercutar el bucle. El control se transferirá a la instrucción siguiente, situada dentro del bucle.

De esta forma, se van ejecutando las instrucciones del mismo, hasta llegar a la instrucción NEXT.

La instrucción NEXT marca, como ya hemos dicho, el final del bucle. Su función es transferir de nuevo el control a la línea siguiente de la instrucción FOR, para la vuelta siguiente.

Segunda vuelta y siguientes. Se inicia el cuerpo del bucle en la instrucción siguiente del FOR dejando inalterada la variable de control.

Se continúa con todas las instrucciones del bucle hasta alcanzar de nuevo la instrucción NEXT.

Se incrementa el valor de la variable de control, es decir, se suma la unidad al valor actual de la misma.

A continuación, se compara el valor resultante con el valor máximo indicado por la propia instrución FOR, presentándose dos posibles situaciones, tal como en la primera vuelta:

— Valor inferior o igual al máximo: se prosigue la ejecución del bucle.

— Valor superior al máximo: no se ejecuta el bucle, transfiriéndose el control a la instrucción siguiente del NEXT.

De esta forma, se va repitiendo el proceso, hasta que la variable de control alcanza el valor máximo.

Ejecute el programa de ejemplo que está escrito con el FOR y compruebe el funcionamiento indicado. El resultado de la ejecución debe de ser idéntico al descrito en el apartado primero de este capítulo.

La variable de control está incrementada cuando sale del bucle

Al salir del bucle, el valor de la variable de control ha superado el máximo en una unidad. Comprobémoslo con el comando de ejecución inmediata:

```
PRINT I
```

Observamos que escribe un 11.

Tal como hemos visto, la última vez que se ejecuta el bucle la I vale 10. Antes de la instrucción NEXT se suma una unidad a la variable de control: I vale 10, pasa a valer 11. A continuación se compara este valor con el máximo, y como es superior, el control se transfiere fuera del bucle, sin modificar el valor de la I. El valor de ésta, por tanto, es 11.

9.2.4 Diversos casos de bucles FOR

Consideremos en primer lugar el programa siguiente:

```
NEW
10 FOR I = 1 TO 5
20 PRINT I,"HOLA"
30 NEXT I
```

imprime cinco veces la variable de control y la palabra «HOLA».
El resultado de la ejecución es:

```
1   HOLA
2   HOLA
3   HOLA
4   HOLA
5   HOLA
```

Si queremos variar el número de repeticiones, modificaremos la línea 10. Para efectuar 10 vueltas, escribiremos:

```
10 FOR I = 1 TO 10
```

Modificación de los límites superiores e inferiores

con lo que en pantalla aparecerá 10 veces la palabra HOLA, precedida del valor de la I correspondiente.
Se puede modificar también el valor inicial de la variable, escribiendo:

```
10 FOR I = 7 TO 10
```

al ejecutarlo ahora, veremos en pantalla:

```
7   HOLA
8   HOLA
9   HOLA
10  HOLA
```

Debe tenerse en cuenta que si el valor inicial que fijamos ya es superior al final, por ejemplo,

```
FOR I= 10 TO 1
```

el comportamiento puede diferir según el modelo de ordenador del que se trate. La diferencia consiste que en algunos lenguajes BASIC la decisión de salir del bucle se realiza en la instrucción FOR y en otros en la instrucción NEXT. Con el esquema que hemos explicado aquí el bucle se realiza por lo menos una vez ya que la decisión de salir del bucle se toma en la instrucción NEXT. Para conocer el funcionamiento de su máquina en particular, tenga en cuenta lo que se dice en la lección de prácticas.

La variable de control puede utilizarse en los cálculos en el interior del bucle

La variable de control puede utilizarse en el cuerpo del bucle para realizar cualquier tipo de cálculo. Esto permite una fácil realización de programas que construyan tablas. Veamos un ejemplo de construcción de la tabla de los cuadrados de los números del 1 al 5. El resultado a obtener es:

1 al cuadrado es 1

2 al cuadrado es 4

3 al cuadrado es 9

4 al cuadrado es 16

5 al cuadrado es 25

El programa para ello se escribe:

```
NEW
10 FOR I = 1 TO 5
20     PRINT I;" al cuadrado es ";I^2
30     NEXT I
```

Ejecutémoslo para comprobar su correcto funcionamiento. Cambiando los valores límite de la variable de control, podremos obtener otras tablas de cuadrados. Así, para tener los cuadrados del número 5 al número 10, bastará escribir:

```
FOR I = 5 TO 10
```

Si queremos la tabla completa, del 1 al 10, deberemos escribir:

```
FOR I = 1 TO 10
```

Las variables de un programa pueden interferir en el cálculo de los valores iniciales y finales de la instrucción FOR

Sin embargo, esto nos resulta un poco incómodo, pues cada vez que queremos variar los límites, debemos editar la línea 10.

Vamos a ver que esto no es necesario. El BASIC nos permite utilizar variables para indicar los valores inicial y final de la variable de control. Así, podemos escribir:

```
10 FOR I = P TO U
```

siendo *P* y *U* dos variables que pueden tomar los valores que se desee. Para asignarlos en cada caso, añadiremos dos instrucciones:

```
5 INPUT "PRIMER VALOR",P
7 INPUT "ULTIMO VALOR",U
```

El listado completo del programa será ahora:

```
 5 INPUT "PRIMER VALOR",P
 7 INPUT "ULTIMO VALOR",U
10 FOR I = P TO U
20     PRINT I;" al cuadrado es ";I^2
30     NEXT I
40     GO TO 5
```

Ejecutando ahora el programa, podemos escribir la tabla que deseemos, sin modificarlo.

Por otra parte, nos podría interesar que cada vez la tabla tuviera 10 valores, de forma que pudiéramos verla completa en pantalla. Para ello, bastaría introducir el valor inicial —que interesara que sea variable— y calcular el valor final cada vez. Así, si el valor inicial que se introduce es 1, el final deberá ser 10; si empezamos por el 5, el último valor será el 14, etc. Por tanto, dado el valor inicial, el final se determina sumándole 9.

Para que este cálculo lo efectúe directamente el programa, podríamos sustituir la línea 7 por:

```
7 LET U = P + 9
```

Si P vale 1, el último valor será 10, con lo que nos escribirá la tabla de cuadrados desde 1 a 10, tal como deseábamos.

Sin embargo, el BASIC nos ofrece una nueva posibilidad, y es la de incluir la operación en la misma instrucción FOR. Así, se puede escribir:

```
10 FOR I = P TO P+9
```

De esta forma, el propio programa determina el valor máximo de la variable de control efectuando la operación indicada a cada vuelta.

Podremos borrar la línea 7, con lo que el listado del programa quedará:

```
 5 INPUT "PRIMER VALOR",P
10 FOR I = P TO P+9
20     PRINT I;" al cuadrado es ";I^2
30     NEXT I
40 STOP
```

Al ejecutar el programa, indicando 1 como valor inicial, obtendremos la tabla:

1 al cuadrado es 1

2 al cuadrado es 4

3 al cuadrado es 9

4 al cuadrado es 16

5 al cuadrado es 25

6 al cuadrado es 36

7 al cuadrado es 49

8 al cuadrado es 64

9 al cuadrado es 81

10 al cuadrado es 100

En el cálculo del valor inicial y final de una instrucción FOR se puede utilizar cualquier expresión válida en BASIC.

En este caso, se asigna al valor final de la variable de control el resultado de una suma. En general, puede hacerse con cualquier operación, y tanto para calcular el valor inicial como el final.

Así, se podría escribir también

```
10 FOR I = P*2 TO P*5
```

Dando 10 como valor de P, tendríamos la tabla de cuadrados de los números 20 al 50. En general, se puede utilizar como valor inicial y final cualquier expresión matemática que cumpla las normas del BASIC.

9.3 BUCLE ESCALONADO

Podría interesarnos obtener únicamente la tabla de los números pares, o de los números de 3 en 3, de 4 en 4...

El BASIC nos permite indicar, dentro de la misma instrucción FOR, un incremento de la variable de control distinto de la unidad. Para ello se escribe la instrucción FOR tal como la conocemos, pero se indica al final de la misma el incremento que se desea para la variable *1*. Esto se hace escribiendo STEP seguido del incremento. STEP significa en inglés *escalón* o *paso*. Así, la instrucción FOR completa se escribe:

FOR variable = primer valor TO último valor STEP intervalo

Para el caso que deseáramos obtener la tabla de los cuadrados de los diez primeros números pares, deberíamos borrar la línea 5, y escribir:

```
10 FOR I = 2 TO 10 STEP 2
```

Ejecutando ahora el programa, el resultado sería

2 al cuadrado es 4

4 al cuadrado es 16

6 al cuadrado es 36

8 al cuadrado es 64

10 al cuadrado es 100

El funcionamiento del bucle no ha variado; se asigna a la variable de control el primer valor (en este caso 2), se ejecuta la primera vuelta, y al calcular el siguiente valor de *I*, en lugar de sumársele la unidad, se le suma el valor del intervalo indicado por el STEP. En este caso será 2+2, por tanto 4. Así se va ejecutando el bucle, *I* va tomando los valores 2, 4, 6, 8 y 10.

El incremento de la variable de control se realiza mediante el valor del paso

El bucle se detiene en el momento que la variable de control alcanza un valor superior al máximo. En este caso, *I* llega al valor 10, y se ejecuta el bucle; a la siguiente vuelta, la *I* valdrá

```
10+2 = 12
```

que es superior al máximo. Por tanto, el control será transferido ya fuera del bucle.

Modifiquemos ahora el programa de forma que el intervalo sea impar (3 por ejemplo). Para ello escribiremos:

```
10 FOR I = 2 TO 10 STEP 3
```

Si ejecutamos ahora el programa, veremos en pantalla:

2 al cuadrado es 4

5 al cuadrado es 25

8 al cuadrado es 64

En este caso, el bucle da tres vueltas, tomando la *I* los valores 2, 5 y 8 sucesivamente. Nunca llega la *I* a igualarse al valor máximo. Sin embargo, el bucle se detiene, porque en la cuarta vuelta la *I* valdrá:

$$8+3=11$$

que ya es superior a 10. En este caso, pues, el bucle no se ejecuta.

El valor del intervalo debe especificarse únicamente si es distinto de la unidad. Si la variable de control debe ir creciendo de 1 en 1 (tal como habíamos visto en secciones anteriores), puede omitirse la indicación del paso (no hace falta escribir STEP 1). Así, la instrucción

```
FOR I = 1 TO 10 STEP 1
```

tiene exactamente el mismo efecto que la instrucción

```
FOR I = 1 TO 10
```

Por otra parte, el BASIC permite indicar el valor del incremento mediante un número o una expresión. Así, si deseamos escribir la tabla de cuadrados variando cada vez el incremento entre los números, podremos escribir:

```
5 INPUT "ENTRE EL INCREMENTO",S
```

y modificar la línea 10 de la siguiente forma:

```
10 FOR I = 1 TO 10 STEP S
```

el listado completo quedará ahora:

```
5 INPUT "ENTRE EL INTERVALO",S
10 FOR I = 1 TO 10 STEP S
20    PRINT I;" al cuadrado es ";I^2
30    NEXT I
```

Ejecutando ahora el programa, podremos observar los resultados distintos que se obtienen al introducir los diferentes valores del intervalo: 1, 2, 3, ...

El paso en el bucle debe ser distinto de cero

Ejecute una vez más el programa, y entre el 0 como valor del intervalo. Observará que el programa queda encallado, escribiendo repetidamente:

1 al cuadrado es 1

Para detenerlo utilizaremos la tecla de interrupción. Más adelante explicaremos qué es lo que ha sucedido; el porqué de que el programa se encalle en un bucle con STEP 0.

9.4 BUCLES DECRECIENTES

Hasta ahora, hemos utilizado únicamente bucles ascendentes, en que el valor de la variable de control crece, el valor inicial es inferior al final. ¿Puede darse la situación inversa? Es decir, ¿se puede construir un bucle en que la variable vaya disminuyendo, desde un valor superior a uno inferior?

O lo que es lo mismo, ¿puede escribirse un bucle decreciente?

Por ejemplo, ¿podríamos utilizar un bucle FOR/NEXT para realizar un programa que escribiera los números pares de 10 hasta 0?

Esto puede hacerse en BASIC utilizando las instrucciones FOR y NEXT tal como ya sabemos, pero indicando un valor negativo del incremento. En este caso, además, es necesario que el valor inicial de la variable de control sea superior al valor final.

Así, el programa para tener los números pares en orden decreciente, se podría escribir:

```
NEW
10 REM Bucle descendente
20 FOR I = 10 TO 0 STEP -2
30    PRINT I
40    NEXT I
50 STOP
```

Al ejecutar el programa, veremos en pantalla la lista de números 10, 8, 6, 4, 2, 0.

El valor de un paso en un bucle puede ser negativo

El funcionamiento de estos bucles es el descrito para los bucles crecientes. Se asigna el primer valor a la variable de control —que en este caso ha de ser superior al valor final de la misma— y se ejecuta el bucle. En la siguiente vuelta, se le sumará al valor actual de la variable control, el valor del STEP. En este caso, será

$$10 + (-2) = 8$$

y proseguirá la ejecución del bucle. Este finalizará después de escribir el 0, ya que al sumar:

$$0 + (-2) = -2$$

obtendremos un valor de *I* inferior al final, con lo que acabará el proceso.

Las características que determinan un bucle decreciente son, pues, que el valor inicial de la variable de control sea superior al valor final, y que el incremento o paso sea negativo.

Cuando el valor final es mayor que el valor inicial, por ejemplo,

```
FOR I = 1 TO 10 STEP -1
```

(este caso se corresponde con el caso del valor final menor que el inicial en el caso de bucles crecientes), el comportamiento del bucle depende de cada ordenador y en el capítulo de prácticas obtendrá el comportamiento exacto para su ordenador. Sin embargo, en el estilo del bucle que se ha explicado aquí, en que la pregunta se realiza en la instrucción NEXT, el bucle se ejecuta por lo menos una vez.

RESUMEN

Es frecuente encontrar en las aplicaciones de la programación estructuras que se repiten. Al mecanismo que permite esta repetición se le denomina bucle o estructura de repetición.

El mecanismo se basa en una variable, que se denomina variable de control.

Esta variable se modifica en cada vuelta o repetición y mediante un mecanismo de decisión, es decir, una instrucción IF...THEN repetimos o no el proceso.

También es necesario un proceso de inicialización antes de empezar el bucle, para dar a la variable de control el valor inicial adecuado.

El BASIC dispone de dos instrucciones para realizar este mecanismo de una manera más cómoda, que son: el FOR y el NEXT.

La instrucción FOR abre el bucle, del mismo modo que se abre un paréntesis y realiza las tareas de inicializar las variables.

La instrucción NEXT incrementa el valor de la variable de control y decide si hay que continuar o no la repetición. Es la instrucción que cierra el paréntesis abierto por la instrucción FOR.

Este mecanismo de coordinación nos indica que no tiene sentido la instrucción FOR sin la NEXT y viceversa.

Las funciones principales de la instrucción FOR consisten en inicializar el valor de la variable de control, calcular el valor final y establecer el paso o escalón del bucle.

Este cálculo se realiza porque pueden darse expresiones en los valores inicial, final y paso que sean válidas en BASIC.

El paso negativo da origen a un bucle decreciente, mientras que un paso positivo da origen a un bucle creciente.

El comportamiento del BASIC si un valor final es menor que el valor inicial en un bucle creciente o si un valor es mayor que el inicial en un bucle decreciente, depende del ordenador que se utilice. Por lo tanto, delante de una máquina desconocida hay que experimentar para poder saber exactamente cuál es el mecanismo de actuación.

Las funciones principales del NEXT, son incrementar la variable de control, según indica el valor del escalón, y comparar este valor con el valor final, para decidir si continuar o no el cuerpo del bucle.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

1. Para programar cosas repetitivas se utiliza el ....
2. El control de la repetición se hace mediante la ....
3. Los bucles se pueden programar sin .... especiales.
4. El BASIC dispone de las instrucciones .... y .... para hacer más sencilla la programación de los bucles.
5. La instrucción FOR marca el .... de un bucle.
6. La instrucción NEXT marca el .... de un bucle.
7. No tiene sentido una instrucción FOR sin una instrucción .... y viceversa.

8. Los valores inicial, final y el paso de un bucle, que se especifican en un FOR pueden ser cualquier .... válida en BASIC.

9. La misión principal de un NEXT es .... la variable de control.

10. Cuando la variable de control disminuye el bucle se denomina ....

Encierre en un círculo la alternativa que corresponde a la respuesta correcta

11. En el bucle

```
FOR I = 1 TO 3 STEP 2
    PRINT I
    NEXT I
```

qué secuencia de números imprime.

a) 1

b) 1, 3

c) 3, 1

d) 3

12. En el bucle

```
FOR I = -1 TO 3 STEP 2
    PRINT I
    NEXT I
```

qué secuencia de números imprime.

a) –1, –3

b) –1, 1, 3

c) 0, 2

d) –2, –4

13. En el bucle

```
FOR I = 9 TO 5 STEP -3
    PRINT I
    NEXT I
```

qué secuencia de números imprime.

a) 9, 6

b) 9, 6, 3

c) 9

d) 6, 3

14. En el bucle

```
FOR I = -3 TO -8 STEP -3
    PRINT I
    NEXT I
```

qué secuencia de números imprime.

a) –3

b) –3, –6, –9

c) –3, –5

d) –3, –6

15. En el bucle

```
FOR I = 101 TO 125 STEP 28
    PRINT I
    NEXT I
```

qué secuencia de números imprime.

a) 101, 112

b) 125

c) 101

d) 128

16. En el bucle

```
LET K = 0
FOR I = 1 TO 5
    LET K = K + I
    PRINT K
    NEXT I
```

qué secuencia de números imprime.

a) 1, 2, 3, 4, 5

b) 5, 6, 7, 8, 9

c) 2, 4, 8, 16, 32

d) 1, 3, 6, 10, 15

17. En el bucle

```
LET K = 0
FOR I = 1 TO 3
    PRINT K
    LET K =(K-I)/2
    NEXT I
```

qué secuencia de números imprime.

a) 0, –2, –4

b) 0, –0.5, –1.25

c) –0.25, –2.125, –4.0625

d) –0.5, –0.25, –0.125

18. En el bucle

```
LET K = 1
FOR I = 1 TO 5
    LET K = K*I
    PRINT K
    NEXT I
```

qué secuencia de números imprime.

a) 1, 12, 72, 86, 234

b) 1, 2, 6, 25, 100

c) 1, 2, 6, 24, 120

d) 1, 2, 6, 26, 140

19. En el bucle

```
LET K = 1
FOR I = 1 TO 5
          PRINT K
          LET K=(K+I)/2+1
          NEXT I
```

qué secuencia de números imprime.

a) 5, 67, 456, 8769, 98356

b) 1, 2, 3, 4, 5

c) 1, 2, 3, 10, 20

d) 0.5, 0.25, 0.125, 0.0625, 0.03125

20. En el bucle

```
LET K = 1
FOR I = 1 TO 5
          PRINT K
          LET K= 2*I-1
          NEXT I
```

qué secuencia de números imprime.

a) 1, 2, 3, 4, 5

b) 1, 3, 5, 7, 9

c) 1, 1, 3, 5, 7

d) 1, 3, 5, 7, 7

Le aconsejamos que compruebe sus respuestas con el ordenador.

9.5 BUCLES ANIDADOS

Veamos otro ejemplo de utilización de un bucle. Se desea escribir la tabla de multiplicar, por ejemplo, la tabla del 5.

El resultado a obtener es:

$5 \times 0 = 0$

$5 \times 1 = 5$

$5 \times 2 = 10$

$5 \times 3 = 15$

$5 \times 4 = 20$

$5 \times 5 = 25$

$5 \times 6 = 30$

$5 \times 7 = 35$

$5 \times 8 = 40$

$5 \times 9 = 45$

$5 \times 10 = 50$

El programa para ello sería:

```
10 REM Tabla de multiplicar
20 FOR I = 0 TO 10
30     PRINT 5;" x ";I;" = ";5*I
40     NEXT I
```

Si deseamos obtener la tabla para otro número cualquiera, modificaríamos el programa añadiendo la línea:

```
15 INPUT "NUMERO: ",N
```

y modificando la 30:

```
30 PRINT N;" x ";I;" = ";N*I
```

El listado completo del programa será ahora:

```
10 REM Tabla de multiplicar
15 INPUT "NUMERO: ",N
```

```
20 FOR I = 0 TO 10
30     PRINT N;" x ";I;" = ";N*I
40     NEXT I
```

De esta forma podríamos obtener la tabla de multiplicar de cualquier número.

Supongamos ahora que nos interesa la tabla de multiplicar de los números del 0 al 10. Podríamos obtenerlas ejecutando sucesivamente el programa y dando cada vez el valor de *N* correspondiente, entre 0 y 10. Pero dado que sabemos los límites entre los que varía *N*, y que lo hace de 1 en 1, podemos añadir un nuevo bucle, que nos gobierne esta variación de *N*. Nos interesaría algo así:

FOR N = 0 TO 10

Escribir la tabla de multiplicar
para el valor actual de N

NEXT N

El programa para escribir la tabla de multiplicar de *N* es el que tenemos escrito. Bastará ahora añadir dos líneas:

```
15 FOR N = 0 TO 10
50 NEXT N
```

El listado completo del programa quedará ahora:

```
  10 REM Tabla de multiplicar
┌─15 FOR N = 0 TO 10
│┌20   FOR I = 0 TO 10
││30     PRINT N;" x ";I;" = ";N*I
│└40     NEXT I
└─50   NEXT N
```

Tal como se señala, tenemos un bucle «anidado» dentro de otro.

El funcionamiento no varía respecto al que ya conocemos. Empieza la ejecución para el primer valor de N (0). Se ejecuta el bucle interior completo (para I de 0 a 10), tal como ya sabemos. Una vez finalizado este bucle, el control salta fuera del bucle interior, a la línea siguiente al mismo. Esta es la línea 50, que al ser la de fin del bucle exterior, transfiere el control de nuevo al principio del bucle (línea 15), con lo que N toma el siguiente valor (N=1), con lo que se ejecutará el bucle interior, y así sucesivamente hasta llegar a N=10. De esta forma obtenemos las diez tablas. Para obtenerlas escritas por separado, añadiremos una instrucción que nos borre la pantalla para cada nueva tabla.

Así escribiremos:

```
17 CLS
```

Esto dejará la pantalla limpia y preparada para escribir la tabla siguiente.

Sin embargo, no acaban aquí las posibilidades. Supongamos que, además, queremos repetir el proceso de escritura de las tablas de todos los números tres veces. Podremos ejecutar tres veces el programa que tenemos, o bien añadir las líneas:

```
12 FOR V = 1 TO 3
60    NEXT V
```

El nivel de anidación de los bucles puede ser cualquiera

Tenemos ahora tres bucles anidados. El programa completo sería:

```
10 REM Tabla de multiplicar
12 FOR V = 1 TO 3
15   FOR N = 0 TO 10
17      CLS
20      FOR I = 0 TO 10
30         PRINT N;" x ";I;" = ";N*I
40         NEXT I
50      NEXT N
60   NEXT V
```

Así pues, repetimos tres veces el proceso de forma automática.

Debe prestarse mucha atención a la correcta colocación de las instrucciones de fin de bucle, ya que no se puede cerrar un bucle más exterior hasta que no haya completado el más interior.

Por eso observe que la línea 40 cierra el bucle que se inicia en la línea 20, la línea 50 cierra el bucle que se inicia en la línea 15, y finalmente la línea 60 cierra el que inicia en la línea 12.

El hecho que la instrucción NEXT lleve la variable de control del bucle es para establecer un mecanismo de comprobación interno del BASIC que sabe así qué bucle deseamos cerrar.

9.6 BUCLES DE ESPERA

Escribamos el programa:

```
NEW
10 FOR I = 1 TO 100
20    NEXT I
```

y ejecutémoslo.

¿Qué es lo que observamos?

En la pantalla no aparece nada. Sin embargo, desde el momento en que hemos escrito RUN, hasta que la máquina da de nuevo la indicación de que está preparada para recibir nuevas órdenes, transcurre un tiempo.

Si cambiamos la línea 10 por:

```
10 FOR I = 1 TO 1000
```

y ejecutamos de nuevo el programa, observamos que el intervalo de tiempo transcurrido es mayor que en el caso anterior.

¿A qué se debe esto?

Observemos en primer lugar el programa que hemos escrito. Veremos que tenemos un bucle, que empieza en la línea 10 y termina en la 20, sin ninguna instrucción en su interior. ¿Por qué pues se entretiene la máquina cuando en realidad el bucle está vacío?

La explicación la encontramos en la forma de funcionamiento interno de los bucles.

Como ya hemos visto, en la instrucción FOR se asigna el valor que le corresponda a la variable, y se va incrementando sucesivamente el valor de la misma, comprobando al mismo tiempo que ésta se encuentre entre los márgenes fijados en la propia instrucción. Por otra parte, la instrucción NEXT nos remite de nuevo a la instrucción FOR.

Cuando se repiten muchas veces instrucciones muy cortas el tiempo se hace apreciable

Evidentemente, la máquina consume un tiempo en dar cada uno de estos pasos. Este tiempo es muy pequeño (prácticamente inapreciable), si ha de efectuarlo una sola vez, pero si lo hacemos repetir 100, 1000 veces, evidentemente multiplicamos el tiempo por 100, o por 1000... Este es el intervalo de espera que apreciamos entre el momento en que escribimos RUN, y el momento en que la máquina vuelve a responder.

Estos bucles, que lo único que hacen es «entretener» a la máquina, se denominan bucles de espera, y tiene utilidades muy diversas. Pueden utilizarse por ejemplo en programas autoexplicativos.

Estos programas incluyen la explicación del funcionamiento, es decir, dan las instrucciones para utilizarlos, al inicio de su ejecución, o durante la misma.

Estas instrucciones aparecen generalmente escritas en pantalla durante un especio de tiempo más o menos largo, para que el usuario tenga tiempo de leerlas.

Veamos un ejemplo de utilización de un bucle de espera para este caso. Escribamos el programa:

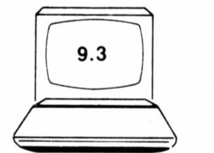
```
NEW
10 CLS
20 PRINT "... este mensaje"
30 PRINT "se autodestruira"
40 PRINT "en 5 segundos..."
50 FOR I = 1 TO 100
60    NEXT I
70 CLS
```

Ejecutando el programa, observaremos que el mensaje aparece en pantalla durante un tiempo, y es borrado automáticamente. Para ajustar exactamente el tiempo a 5 segundos, puede modificar la línea 50, variando el límite superior de la variable de control (100), por otros valores (150, 200, ...) hasta que el mensaje permanezca exactamente 5 segundos en pantalla antes de desaparecer.

Otro ejemplo de utilización de estos bucles de espera lo podemos ver si construimos un programa que simule un reloj. Este programa nos servirá también como ejemplo de utilización de bucles anidados. El programa para ello será:

```
NEW
10 FOR I = 0 TO 23
20      FOR J = 0 TO 59
30          FOR K = 0 TO 59
40              CLS
50              PRINT I;":";J;":";K;
60              FOR L = 1 TO 100
70                  NEXT L
80              NEXT K
90          NEXT J
100     NEXT I
```

Ejecutando el programa, veremos que aparece la hora completa, escrita segundo a segundo, en la parte superior izquierda de la pantalla. Como en el caso anterior, ajuste el tiempo modificando de forma adecuada el valor final de la variable de control del bucle de espera.



9.7 EL GOTO Y LOS BUCLES

Una de las instrucciones que disponemos es el GOTO que nos permite saltar de un sitio a otro del programa. Por otra parte, como ya hemos comprobado, las instrucciones FOR y NEXT actúan de una manera coordinada pero son instrucciones distintas. Entre ellas podemos colocar muchísimas instrucciones.

Por lo tanto, en principio podemos utilizar instrucciones GOTO, y con ellas saltarnos el mecanismo de funcionamiento del bucle. La utilización del GOTO permite entrar y salir dentro de los bucles a nuestro antojo.

Sin embargo, si la utilización del GOTO para estos menesteres no está cuidadosamente preparada, esta utilización del GOTO nos puede llevar a desastres en el comportamiento del programa. Si, además, esto se hace a la ligera, la dificultad de encontrar la causa puede costarnos mucho tiempo, con el consecuente desánimo.

En esta sección vamos a analizar cómo debe hacerse para utilizar el GOTO para modificar el mecanismo de las instrucciones FOR y NEXT.

9.7.1 Salida de los bucles

Consideremos, en primer lugar, la utilización del GOTO para la salida de los bucles.

Como norma general, siempre es posible utilizar un GOTO para salir de un bucle sin peligro alguno de que el funcionamiento sea incorrecto. En otras palabras, nos podemos saltar una instrucción NEXT sin peligro.

Diremos además que suele ser frecuente encontrar programas que utilizan esta manera de funcionar.

¿Cuándo se debe utilizar este tipo de salida? Generalmente nos encontramos que hay procesos que tienen un límite máximo, pero que si se da cierta condición se puede finalizar antes.

Procesos que pueden finalizar antes del tope máximo previsto

Por ejemplo, cuando Ud. busca el nombre de un amigo en su agenda, empieza a repasar desde el primer nombre (quizá desde el primer nombre en la letra correspondiente de su índice). En el momento que encuentra el nombre, finaliza el proceso de búsqueda. De todas maneras finalizará la búsqueda si llega al final de la lista y ve que no tiene el nombre de su amigo en la agenda; en estos momentos quizá lo apuntará o hará cualquier otra acción.

El mecanismo que nos interesa resaltar en el ejemplo es: se inicia un proceso repetitivo: buscar el nombre. Este proceso se repetirá como máximo la longitud de la lista de la agenda o se acabará antes, en el momento en que ha encontrado el nombre que buscaba.

Veamos ahora un ejemplo con un programa. Considere que quiere realizar un juego con el ordenador de tal manera que el jugador debe acertar un número que el ordenador ha memorizado en menos de diez jugadas. El programa es el siguiente:

```
  10 LET N = 13
  20 FOR I = 1 TO 10
  30   PRINT "Jugada: ";I;" elija un numero:";
  40   INPUT A
  50   IF A = N THEN GOTO 100
  60   PRINT "Numero equivocado"
  70   NEXT I
  80   PRINT "Lo siento no ha acertado el numero"
  90   GOTO 1000
 100 PRINT "Ha acertado el numero. Muy bien"
1000 STOP
```

El programa consiste en:

1. El ordenador elige el número 13 que coloca en la variable *N* (línea 10). En lecciones posteriores veremos cómo se puede escoger un número al azar.

2. El programa entra en un bucle que se inicia en la línea 20 y acaba en la línea 70, este bucle se repetirá 10 veces que es el número de oportunidades que damos al jugador. El cuerpo del bucle consiste en informar al jugador de la jugada que está realizando y le pide un número. (Líneas 30 y 40.)

3. La instrucción de la línea 50 es una decisión que indica que si el número se ha acertado, se sale fuera del bucle. En caso contrario seguirá en el cuerpo del bucle informando al jugador que el número es equivocado.

4. Si se alcanza la jugada diez y el jugador no ha acertado el número, el bucle finaliza y se informa al jugador (línea 80) de que no ha conseguido el objetivo.

5. Cuando nos salimos del bucle mediante el GOTO a la instrucción 100, se informa al jugador que ha acertado el número. Teclee ahora el RUN y compruebe el funcionamiento del programa introduciendo en una primera fase números distintos a 13 para provocar la repetición de las 10 jugadas.

En una segunda prueba teclee el trece en un momento determinado y compruebe que sale del bucle.

La instrucción GOTO puede saltarse instrucciones NEXT sin peligro

Por lo tanto, siempre es posible saltarse la instrucción NEXT mediante un GOTO; de todas maneras, procure no abusar de esta construcción. Si hay demasiados saltos fuera de los bucles se producen dificultades para seguir el programa y puede generar un caos. En conclusión, sea prudente al realizar este tipo de construcciones que, aunque permitidas, debe restringirlas a las aplicaciones estrictamente necesarias.

9.7.2 Entrada de los bucles

Como norma general *no es posible realizar saltos al interior de un bucle.*

La regla es que un GOTO no puede traspasar una instrucción FOR, si no traspasa también la correspondiente instrucción NEXT.

El BASIC detecta un error cuando queremos realizar una operación de este tipo. El error se detecta al alcanzar la instrucción NEXT. Todos los BASIC suelen dar un mensaje como NEXT sin FOR o algo parecido.

De acuerdo con lo que hemos dicho al principio del capítulo, es lógico el mensaje, pues la máquina detecta que deseamos cerrar un paréntesis que previamente no hemos abierto.

Considere el programa que escribe 10 veces Hola, pero en el que se ha añadido una instrucción para entrar en el interior del FOR sin pasar por él.

```
10 GOTO 30
20 FOR I= 1 TO 10
30    PRINT I,"HOLA"
40    NEXT I
50 PRINT "He acabado"
```

Al ejecutarlo, la línea 10 nos transfiere el control a la instrucción 30 sin pasar antes por la instrucción 20 que es el FOR.

Al llegar a la instrucción NEXT se producirá el error que dirá más o menos que ha encontrado un NEXT para el que no se ha abierto el FOR correspondiente.

Es correcto traspasar un FOR si también se traspasa el NEXT correspondiente

En cambio no detectaremos error, porque no lo es, si traspasamos la estructura entera del FOR y el NEXT. Si modifica la instrucción 10 y la cambia a

```
GOTO 50
```

el programa escribirá que ha acabado sin mayor problemas pues nos hemos saltado la estructura completa.

Esta forma de realizar el salto se puede decir que incluso es frecuente. Típicamente en el FOR que hemos explicado aquí cuando en el bucle creciente el valor final es menor que el valor inicial y a la inversa en el decreciente el bucle se ejecuta una vez por lo menos. Sin embargo, nuestro deseo es que no lo realice ninguna vez, entonces no tenemos más remedio que colocar un IF previo para saltarnos la estructura completa si el bucle no debe ejecutarse ninguna vez. Tenemos el programa que calcula la tabla de cuadrados del apartado 2.4.

```
 5 INPUT "PRIMER VALOR",P
 7 INPUT "ULTIMO VALOR",U
 9 IF (P > U ) THEN GOTO 40
10 FOR I = P TO U
20     PRINT I;" al cuadrado es ";I^2
30     NEXT I
40     GO TO 5
```

Al programa se le ha añadido la instrucción 9 que es una instrucción IF que se salta el bucle entero si el valor inicial es mayor que el valor final.

Seguramente se preguntará por qué no lo envía directamente a la instrucción 5. Ciertamente tiene razón, pero se ha utilizado como ejemplo sencillo, y en otros casos las instrucciones que siguen al bucle pueden tener sentido realizarlas aunque el valor inicial sea mayor que el final.

En las máquinas que no tienen este comportamiento el IF no es necesario en este caso, pero la regla de que se puede traspasar un FOR y un NEXT con un GOTO es aún válida.

Hay ejemplos más complicados de utilización correcta de la entrada dentro de un bucle, pero en cualquier caso son desaconsejables.

Considere un programa para escribir en dos columnas. Para hacerlo más sencillo escribiremos el mismo texto en las dos columnas.

El proceso de escritura es un proceso repetitivo, pero que debe hacer acciones distintas cuando está en un valor de la variable de control par o impar. Para tener en cuenta esta diferencia utilizaremos una variable lógica, de tal manera que vaya tomando los valores verdadero y falso, en concordancia con la escritura a la derecha y a la izquierda. La variable lógica debe responder la pregunta ¿debo escribir a la derecha?

Una versión posible es la siguiente:

```
10 LET T$="*******"
20 LET R = 0
```

```
 30 FOR I = 1 TO 10
 40 GOTO 500
 50 LET R = NOT R
 60 NEXT I
 70 STOP
500 IF (R)  THEN PRINT T$
510 IF (NOT R) THEN PRINT T$,
520 GOTO 50
```

Las características del programa son:

1. La variable T$ es el texto a imprimir y contiene 7 asteriscos que se asignan en la línea 10.

2. La variable R es la variable lógica que indica si quiero imprimir a la derecha. Se inicia en la línea 20 y se coloca a falso pues se comienza la impresión a la izquierda.

3. El bucle comprendido entre las instrucciones 30 y 60 es el proceso repetitivo de escritura de 10 veces el texto, y cuya variable de control es la I.

4. El cuerpo del bucle consiste en una instrucción de salida del bucle, la línea 40, que nos transfiere el control a la línea 500. Naturalmente la salida del bucle es totalmente permitida porque sólo atraviesa una instrucción NEXT.

5. La línea 50 es la instrucción que va cambiando el valor de la variable lógica de derecha a izquierda, cambiando el valor de verdadero a falso sucesivamente.

6. La línea 70 finaliza el programa.

7. La línea 500 nos escribe el caso de que sea a la derecha y la línea 510 nos escribe el caso de la izquierda. Observe que sólo se puede ejecutar una de las instrucciones 500 ó 510 pues el valor de la variable lógica y el de su negación no pueden ser verdaderos simultáneamente. También hay que notar que las dos instrucciones PRINT difieren ligeramente. Por eso hay que hacerlas separadas. En el caso de la escritura de la izquierda, línea 510, el PRINT acaba con una coma para dejar el cursor en la parte izquierda de la pantalla, mientras el PRINT de la derecha, línea 500, no lleva ningún signo de puntuación para que escriba en la línea siguiente.

8. La línea 520 transfiere el programa otra vez al interior del bucle. Esta es la instrucción que parece violar la regla de que no se puede entrar en el interior de un bucle.

Ejecutaremos el programa en primer lugar, antes de analizar por qué la instrucción 520 es correcta.

Verá que cuando teclea el RUN el programa transcurre según los planes previstos a pesar de que la instrucción 520 no debe estar permitida.

La razón de que no sea incorrecta la instrucción 520 es que de hecho volvemos a entrar en un bucle que se ha inicializado correctamente con la

instrucción FOR, después nos hemos salido (línea 40) y hemos vuelto a entrar en la línea 50.

Por lo tanto, no se cumple el supuesto de que intentamos pasar por un NEXT (cerrar el paréntesis) sin haber pasado previamente por un FOR (abrir el paréntesis).

Es posible salir y volver a entrar en un bucle

Ciertamente no es una práctica recomendable utilizar este tipo de transferencia, pero el mecanismo está permitido si se planifica cuidadosamente, ya que no viola la regla de atravesar un FOR.

En nuestra opinión es un programa mucho más claro el siguiente:

```
10 LET T$="*******"
20 LET R = 0
30 FOR I = 1 TO 10
40 IF(R) THEN PRINT T$.
50 IF (NOT R) THEN PRINT T$,
60 LET R = NOT R
70 NEXT I
```

También es cierto que este programa es muy sencillo, en programas más complicados este modo de programar puede estar justificado. Tanto es así, que en el capítulo 11, en el tercer tomo, estudiaremos una instrucción que permite realizar este tipo de transferencias de una manera mucho más elegante, clara y efectiva.

Obviamente todas estas reglas deben cumplirse también si los bucles están anidados. En estos casos es mucho más complicado analizar todas las posibilidades. Sin embargo, las normas deben cumplirse para cada uno de los bucles, estén anidados o no. En otras palabras, el anidamiento sólo nos multiplica el número de bucles a controlar, pero no añade ninguna regla nueva respecto a las que hemos visto.

9.8 POSIBILIDADES DE LOS BUCLES FOR/NEXT

La utilización de las instrucciones FOR y NEXT para el control de bucles, nos ofrece, como ya hemos visto, unas amplias posibilidades. Sin embargo, debe ponerse atención para utilizarlos correctamente.

Vamos a repasar brevemente las posibilidades y limitaciones que ofrece la utilización de estas instrucciones.

La forma general de escribir estos bucles, es, como ya hemos visto, mediante dos instrucciones: la que inicia el bucle,

FOR vc = exi TO exf STEP exp

En la que:

vc, se refiere a la variable de control.
exi, se refiere a una expresión en BASIC que da el valor inicial.
exf, se refiere a la expresión que da el valor final.
exp, se refiere a la expresión que da el paso o escalón.

Y la que lo finaliza

NEXT variable de control

9.8.1 Propiedades generales

1. Los valores inicial, final y el incremento pueden ser números, variables, o expresiones, siempre que éstas sean correctas en BASIC.

2. Los bucles pueden ser crecientes o decrecientes. Para escribir un bucle creciente se deberá tener el valor inicial inferior al valor final, y el incremento ha de ser positivo. Para escribir un bucle decreciente se debe tener el valor inicial superior al final, y el incremento debe ser negativo.

3. Dentro de un bucle pueden abrirse otros bucles, es decir, se pueden tener bucles «anidados».

4. Es posible atravesar un NEXT con una instrucción GOTO.

5. No se puede entrar en el interior de un bucle mediante una instrucción GOTO.

6. Se puede atravesar con un GOTO un FOR y un NEXT a la vez.

7. Se puede salir del interior de un bucle y posteriormente entrar en este mismo bucle.

Los bucles FOR/NEXT ofrecen muchas facilidades, pero debe ponerse mucha atención al utilizarlos, para que el bucle haga exactamente lo que deseamos.

9.8.2 Utilización peligrosa del bucle

Vamos a exponer una serie de cuestiones que no son errores, es decir, no están prohibidas, pero son desaconsejables, pues pueden inducir a errores en el programa, y son difíciles de detectar.

1. La variable de control no debe modificarse a lo largo del cuerpo del bucle.

La variable de control puede ser utilizada en el cuerpo del bucle en cualquier sentencia de BASIC. Entre las sentencias está la asignación de la variable de control a otro valor.

No hay que modificar la variable de control

Veamos a continuación dos ejemplos de esta utilización que sin ser incorrecta puede traer problemas.

a) Consideremos el programa siguiente:

```
10 FOR I = 1 TO 5
```

```
20        LET I = I-1
30        PRINT I
40        NEXT I
```

La línea 20 utiliza una instrucción de asignación que modifica la variable de control I, restándole una unidad cada vez.

Vamos a seguir el programa paso a paso.

La instrucción 10 es el comienzo de un bucle. En ella se asigna el valor inicial a la variable de control del mismo. Por tanto:

$$I = 1$$

En la instrucción siguiente —línea 20— se realiza una operación sobre la variable de control, que consiste en restarle 1. Por tanto:

$$I = 0$$

Llegamos ahora al final del bucle, en la instrucción 40 se le suma la unidad a la variable de control. Por tanto:

$$I = 1$$

y el control se transfiere a la línea 20 pues no se cumple que I sea mayor que 5. Al pasar de nuevo por la línea 20 se le resta una unidad; por consiguiente, I vale de nuevo 0 y es el valor que imprime en el PRINT, al llegar a la línea 40 se incrementará en una unidad y por consiguiente valdrá 1, y así sucesivamente, sin posibilidad ninguna de finalización del bucle.

I se moverá alternativamente entre 0 y 1 sin alcanzar nunca el valor final del bucle. De esta forma, éste se seguirá ejecutando, hasta que forcemos la finalización con la tecla de interrupción.

La utilización de la asignación para variar la variable de control nos ha llevado a un bucle sin fin. Debido a que en el interior del bucle había un PRINT nos hemos enterado de que algo equivocado sucedía, pero si no hay ningún PRINT la máquina se queda aparentemente sin hacer nada. Está en un bucle de espera infinito. Compruebe quitando la línea 30 cuál es el estado de la máquina en este caso.

b) Vamos a mostrar que esto se puede utilizar para un efecto práctico deseado. Consideremos el caso de querer escribir las potencias de dos (2, 4, 9...). Cada número consiste en el anterior multiplicando por dos, hasta el valor que sobrepase el 100.

Una posible versión de este programa puede ser:

```
10 FOR I= 2 TO 100
20 LET I = (I-1)*2
30 PRINT I
40 NEXT I
```

En primer lugar ejecútelo y observe que obtiene la secuencia de 2, 4, 8, 16, 32, 64, 128.

¿Por qué?

El truco radica en la instrucción 20, que vamos a comentar. En la primera vuelta la I empieza en 2, al restarle una unidad y multiplicarlo por 2 nos queda 2.

Imprimimos este valor.

A continuación en el NEXT, incrementamos una unidad la I, es decir ahora valdrá tres y como no supera al 100, ejecutamos de nuevo la línea 20.

En ella restamos una unidad a I, obtenemos 2, y la multiplicamos por dos, con lo que se obtiene 4. Imprimimos el valor.

Llegamos de nuevo al NEXT, incrementamos el valor en 1, ahora I vale 5, como no supera a 100 ejecutamos de nuevo la instrucción 20.

En ella disminuimos en 1 la I, vuelve a valer 4 y la multiplicamos por dos, con lo que se obtiene 8.

El proceso se repite sucesivamente hasta llegar a un valor de I superior a 100 que cuando llegue al NEXT saldrá del bucle.

Efectivamente, después de este análisis cuidadoso vemos que el programa está bien planificado para ejercer su función.

Sin embargo, esta manera de programar es sumamente peligrosa. En primer lugar porque al cabo de un tiempo, nos va a costar mucho entender lo que hace el bucle. En segundo lugar, porque un pequeño despiste puede llevarnos fácilmente a un bucle sin fin.

En efecto, cambie inocentemente la instrucción del FOR y en lugar de empezar en 2 empiece en uno, es decir,

```
10 FOR I = 1 TO 10
```

cae inmediatamente en un bucle sin fin. Observe que ahora en el primer valor de la instrucción 20, obtiene un valor de cero, ya que uno menos uno da cero que multiplicado por dos continúa dando cero.

Al llegar al NEXT incrementa la variable en 1, dará por lo tanto 1, e irá a ejectuar otra vez 20 con el mismo valor de antes, obtendrá un valor de cero y el bucle no se termina jamás.

Es mucho mejor utilizar un programa que no utilice un FOR en este tipo de cálculos, por ejemplo,

```
10 LET I=2
20 LET I = I*2
30 PRINT I
40 IF ( I ( 100) THEN GOTO 20
```

De hecho este esquema es muy parecido a un bucle, pero en el que nosotros controlamos todas las fases mediante instrucciones elementales.

2. No debe utilizarse para calcular el valor inicial, ni el final, ni el incremento la variable que se utiliza para control del bucle.

No debe utilizar los valores de I en el cálculo del valor inicial, final y paso

Veamos tres ejemplos:

a) Utilización de la variable de control para calcular el valor inicial, como en:

```
FOR I = I TO 10
```

de hecho, la variable de control puede ser cualquier variable del programa; por lo tanto, nos puede convenir utilizar el mismo nombre de la variable de control que una variable del programa. Entonces lo que hace es iniciar el bucle con el valor que previamente tenía el programa.

Sin embargo, después de salir del bucle esta variable ha cambiado de valor según las vueltas que ha dado. Esto puede ser un inconveniente.

Una buena política es escoger las variables que van dentro de los bucles con unos nombres típicos, frecuentemente son I, J, K, etc., y sólo utilizarlas como variables de control del bucle. De esta manera, no alteraremos inadvertidamente otras variables del programa.

b) Utilizar la propia variable para el cálculo del valor final.

Como en:

```
FOR I = 1 TO I*2
```

Esta situación sí que es verdaderamente absurda, pues es totalmente equivalente a

```
FOR I = 1 TO 2
```

ya que en el momento de calcular el valor final la I vale 1.

Su única justificación sería que el valor inicial fuera una expresión muy complicada, con lo que se evitaría el cálculo dos veces.

A pesar de esta justificación, no es nada recomendable su uso, es mejor utilizar una variable intermedia como en el caso

```
FOR I = A TO A*2
```

el resultado es equivalente pero mucho más claro.

No es fácil decir si en todos los BASIC, el cálculo del valor final se hace una sola vez o se recalcula toda la expresión cada vez que se hace la pregunta. En este segundo caso el valor final depende del número de vueltas realizado. En el capítulo de prácticas encontrará el funcionamiento detallado en estos casos.

c) Hacer intervenir la variable de control en el cálculo del paso.

```
FOR I = 1 TO 10 STEP I+1
```

Los.comentarios son idénticos a los expresados en el caso anterior.

3. El valor del incremento debe ser distinto de cero, de lo contrario, generamos bucles sin final, tal como hemos comprobado anteriormente. Si el valor del STEP es cero, la variable de control tomará el valor inicial, al sumarle el incremento nulo no variará, con lo que nunca podrá alcanzar el valor máximo, de forma que no podremos salir del bucle. El programa seguirá dando vueltas hasta que lo detengamos mediante la tecla de interrupción.

Debe comprobarse que el paso no tiene el valor cero

Es difícil que coloquemos un valor cero en el paso, cuando colocamos un valor.

Pero, como ya sabe en el cálculo del paso es posible colocar una expresión o una variable, cuyo valor puede ser nulo. Caemos inadvertidamente en un bucle sin fin.

Es muy prudente calcular la expresión antes y comprobar que no nos dé realmente cero, que nos lleva a un bucle sin fin.

RESUMEN

En el interior de un bucle es posible abrir otro bucle y cerrarlo antes de cerrar el más exterior.

A este proceso de meter bucles dentro de otros bucles se denomina *anidamiento*, y la relación que se establece entre los dos bucles, es de *bucles anidados.*

El BASIC utiliza el valor de la variable en el NEXT como un mecanismo de comprobación adicional para averiguar las intenciones que tiene el programador al utilizarla como cierre de un FOR.

Los bucles pueden utilizarse también para el control del tiempo del ordenador. Cuando se repite un bucle que no contiene ninguna instrucción en su cuerpo el tiempo puede hacerse grande colocando valores finales elevados. Muchas instrucciones muy cortas dan un tiempo apreciable.

Programar un reloj es un ejemplo de utilización de bucles de espera.

Mediante la instrucción GOTO podemos entrar y salir del bucle. Esto provoca inconvenientes debido a la coordinación tan intensa que hay entre el FOR y el NEXT.

Es posible atravesar con un GOTO una instrucción NEXT, sin peligro de error.

Se aplica en procesos en que, aparte de un tope máximo de repeticiones, el proceso repetitivo puede finalizar por otras causas.

Por otra parte, no es posible para una instrucción GOTO saltarse un FOR, es decir, meterse en el interior de un bucle.

Una instrucción GOTO puede saltarse un FOR si además se salta el NEXT correspondiente.

Es posible salir del interior de un bucle y volver a entrar, pues no violamos la regla de un salto de un FOR sin su NEXT.

Las propiedades generales de los bucles son:

1. En el cálculo de valores inicial, final y paso pueden utilizarse expresiones.
2. Los bucles pueden ser crecientes o decrecientes.
3. Dentro de un bucle puede abrirse otro bucle.
4. Un GOTO puede atravesar un NEXT.
5. Un GOTO no puede atravesar un FOR solo.
6. Un GOTO puede atravesar un FOR y su NEXT.
7. Es posible salir y entrar en el interior del mismo bucle.

Hay situaciones que, sin ser error, son muy peligrosas en cuanto a claridad del programa.

Estas situaciones se deben evitar y son las siguientes:

1. Modificar la variable de control en el interior de un bucle.

2. Utilizar la variable de control en el cálculo de los valores inicial, final y paso.

3. Utilizar un valor de cero en el paso.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

21. Dentro de un bucle es posible abrir y .... otro bucle.

22. El cuerpo de un bucle puede carecer de ....

23. Un bucle sin instrucciones se utiliza para hacer .... el ordenador.

24. Es posible salirse de un bucle mediante la instrucción ....

25. No es posible entrar en un bucle sin ejecutar la instrucción ....

26. No es aconsejable modificar el valor de la .... en el cuerpo de un bucle.

27. La variable de control puede intervenir en las .... de cálculo de los valores inicial, final y paso.

28. Es posible salir y volver a .... en el cuerpo de un bucle mediante las instrucciones GOTO.

29. Si el paso de un bucle vale .... no salimos jamás del mismo.

30. La instrucción NEXT lleva la variable de control para comprobar las intenciones del programador respecto a qué bucle desea ....

31. A continuación le sugerimos 10 situaciones:

    31.1 Un GOTO atraviesa un NEXT.

    31.2 Un GOTO atraviesa un FOR.

    31.3 Se sale y entra en el bucle.

    31.4 Un GOTO atraviesa un FOR y un NEXT.

    31.5 Se modifica la variable de control en el interior del bucle.

    31.6 Se utiliza el valor del paso que depende de la variable de control.

    31.7 Se utiliza el valor final que depende de la variable de control.

31.8 Es un bucle sin fin.

31.9 Es un bucle correctamente formado, en el sentido de no tener errores y no utilizar construcciones no recomendables.

31.10 La anidación de los bucles es incorrecta.

Seguidamente le ofrecemos 10 programas que presentan alguna de las 10 características anteriores. Ud. debe realizar la asociación entre el programa y el concepto anteriormente.

Advertencia: Hay casos en que puede haber conflicto, así por ejemplo, el concepto, un GOTO atraviesa un FOR y un NEXT, tiene un bucle correcto que podría asociarse con el concepto 31.9, de bucle correcto.

Le aseguramos que el bucle correcto no tiene ninguna de las otras características mencionadas.

a)

```
10 LET I=23:LET J=100
20 FOR I= 1 TO 5*I STEP J/10
30   LET A = I+J
40   NEXT I
```

b)

```
10 INPUT A
20 IF A>5 THEN GOTO 40
30 FOR I= A TO A+7
40   LET A = I+3
50   PRINT A
60   NEXT I
```

c)

```
10 LET J=1
20 FOR I = J TO 6
30   LET J=J*I
40   NEXT I
```

d)

```
10 LET I=6
20 FOR I=1 TO 10 STEP I-1
30   PRINT I
40   NEXT I
```

e)

```
10 LET J=3
20 FOR I= 1 TO 20
30   IF ( J>2) THEN GOTO 20
40   NEXT I
```

f)

```
10 INPUT A
20 IF A<=0 THEN GOTO 60
30 FOR I= 1 TO 5
40    PRINT SQR(A*I)
50    NEXT I
60 GOTO 10
```

g)

```
10 LET I=1
20 FOR I = 1 TO 10
30    LET I = I*I
40    NEXT I
```

h)

```
10 FOR I= 1 TO 10 .
20    FOR J= I TO 10
30       PRINT 10*I+J
40       NEXT I
50    NEXT J
```

i)

```
10 INPUT A:INPUT B
20 FOR I = A TO B
30    IF ( I>2 ) THEN GOTO 50
40    NEXT I
50 PRINT A*B+I
```

j)

```
10 INPUT A
20 FOR I = A TO A-5 STEP -1
30    IF ( I<0) THEN GOTO 70
40    PRINT SQR(I)
50    NEXT I
60 GOTO 10
70 PRINT "No sacar la raiz cuadrada"
80 GOTO 50
```

9.5

Capítulo 10
• Los conjuntos dimensionales

ESQUEMA DE CONTENIDO

Objetivos	
Introducción	Listas y tablas Homogeneidad Nomenclatura
Reserva de memoria	Instrucción DIM Variables textuales Filas y columnas Extensiones de la instrucción DIM Empleo de la instrucción DIM
Utilización de conjuntos dimensionados	Subíndices calculados Subíndices de una tabla Elemento cero
Procedimientos básicos	Llenar un conjunto Escribir un conjunto Máximo y mínimo de una lista Suma de los elementos Localización por número Localización por nombre Fecha en letras Cálculo de la media

10.0 OBJETIVOS

El capítulo anterior nos ha introducido en los bucles. Estos nos permiten programar acciones repetitivas con cierta comodidad. En este capítulo estudiaremos la manera de extender el tratamiento a los datos.

Muchas veces los problemas son conceptualmente idénticos pero se deben realizar sobre datos distintos. Es necesario, por lo tanto, disponer de una manera de organizar grandes cantidades de información con una misma lógica y acceder de una manera ordenada a cada uno de los componentes.

Los ordenadores demuestran su potencia cuando hay que procesar un gran número de datos. En este capítulo estudiaremos el caso de que estos datos sean iguales. Esto puede parecer una limitación importante, pero en realidad no lo es en absoluto y la mayoría de problemas pueden reducirse a versiones en que la estructura de los datos sean iguales.

La característica más sobresaliente de la manipulación de estos datos es que hay que realizar una previsión de la memoria que debemos reservar para las necesidades de nuestro problema.

Hay que informar de esta previsión al ordenador mediante la instrucción DIM para que haga efectiva la reserva de memoria.

Es importante en este punto que entienda bien lo que significa la instrucción DIM. Debe aprender cuál es el significado de conjunto dimensionado y cuántas son las dimensiones que va a utilizar en la resolución de un problema. Enseguida aprenderá lo que quiere decir conjunto dimensionado.

Una vez reservado el espacio de memoria, es necesario acceder a cada uno de los elementos. El método de acceso se hace mediante subíndice. El concepto de subíndice es muy importante para manejar correctamente estas estructuras. También lo veremos enseguida.

Ponga especial atención en distinguir cuál es el subíndice de un elemento y cuál es el contenido del elemento.

La experiencia indica que es un error frecuente confundir la posición con el contenido. Las primeras veces que lo utilice haga un acto de reflexión consciente para delimitar de qué parte se está hablando, si de posición o contenido.Con el tiempo verá que esta distinción es muy sencilla.

Una vez establecidos estos conceptos básicos este capítulo le ayudará a comprender su significado mediante abundantes ejemplos.

Estos procedimientos fundamentales demuestran más la técnica que la potencia de los conjuntos dimensionados. Sin embargo, esta técnica es imprescindible para poder realizar programas de utilidad.

Estudie con detenimiento cada uno de los problemas, la dificultad de los problemas no radica en lo que hacen sino en las ideas que utilizan. Deben poder utilizar estos recursos cuando se plantee un problema nuevo.

Los conjuntos dimensionados tampoco son estrictamente necesarios para la programación, pero es una comodidad que se convierte en necesidad cuando hay que procesar mucha información.

10.1 INTRODUCCION

Hasta el momento hemos manejado variables simples, es decir, que contenían un único dato, que podía ser textual o numérico. Este tipo de

variables es suficiente si la cantidad de información a procesar es reducida. El problema surge cuando hay que manipular simultáneamente una gran cantidad de datos. En este caso tendremos muchísimas variables y se hará difícil recordar qué dato tiene cada una. Incluso, si hay muchos datos y las normas de nomenclatura de las variables son muy restrictivas puede ocurrir que no haya suficientes combinaciones de letras para dar nombre a todas ellas.

Por otra parte, el número de instrucciones del programa crece desmesuradamente. Así, por ejemplo, si hay que escribir los resultados habrá que utilizar instrucciones PRINT para todas las variables. Lo mismo ocurre con las instrucciones LET, INPUT, etc.

10.1.1 Listas y tablas

Para solucionar este problema hay que utilizar un tipo nuevo de variables que permitan almacenar varios datos a la vez. Estas variables se denominan *conjuntos dimensionados.* La palabra conjunto se debe a que cada variable memoriza un conjunto de datos. La palabra *dimensionado* se utiliza por analogía con las tres dimensiones del espacio (largo, ancho y alto). Vamos a ver ahora los principales conjuntos dimensionados que utilizaremos, como son las listas y las tablas.

Una *lista* de datos está formada por una relación de elementos sucesivos. Dicho de otro modo, es un *conjunto* de datos de una dimensión. Por ejemplo, las notas que saca un alumno en cada una de las asignaturas.

En una *tabla,* sin embargo, los datos están dispuestos en filas y columnas. Por tanto, se trata de un conjunto de dos dimensiones. Por ejemplo, las notas que sacan todos los alumnos de una clase en cada una de las asignaturas, donde una de las dimensiones estaría formada por los alumnos y la otra por las asignaturas.

Existen también tablas con tres entradas y constituyen, en consecuencia, un conjunto de tres dimensiones. Lógicamente no se pueden representar tablas tridimensionales sobre papel. En su lugar se emplean procedimientos tales como conjuntos de tablas, conjuntos de columnas, etc. Es posible incluso imaginar tablas de cuatro o más dimensiones, aunque sólo se pueden representar subconjuntos de ellas.

El ordenador no tiene ninguna dificultad en emplear conjuntos de varias dimensiones, aunque en la práctica, las listas y tablas son, con mucho, las más utilizadas.

10.1.2 Homogeneidad

Todos los elementos de los conjuntos deben ser del mismo tipo

Los conjuntos dimensionados pueden contener datos numéricos o textuales. Sin embargo debe cumplirse una condición: todos los elementos del conjunto deben ser del mismo tipo (conjunto homogéneo). Si el conjunto es numérico, todos los elementos serán numéricos. Si el conjunto es textual todos los elementos serán textuales.

Hay que recalcar el hecho de que cada elemento del conjunto es equivalente a una variable simple de las que habíamos estudiado hasta ahora.

Una *lista numérica* se denomina también *vector* y una *tabla numérica* se denomina *matriz.* Estos nombres provienen del Algebra matricial, que son una parte de las Matemáticas.

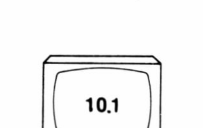
Nosotros trabajamos en general con listas y tablas, pero se pueden emplear directamente los conjuntos dimensionados para el cálculo matricial.

10.1.3 Nomenclatura

Las variables que indican conjuntos dimensionados siguen las mismas normas que las variables simples

En principio se siguen las mismas normas de nomenclatura que para las variables simples, incluyendo el símbolo dólar ($) para los conjuntos que contengan textos.

Sin embargo, algunas versiones del BASIC restringen el nombre de los conjuntos a una sola letra. En el capítulo de «Prácticas con el microordenador» verá enseguida las reglas que se aplican en su ordenador.



10.2 RESERVA DE MEMORIA

Antes de utilizar los conjuntos o variables dimensionadas hay que establecer sus dimensiones a fin de que el BASIC efectúe la correspondiente reserva de memoria. Este comportamiento es distinto al de las variables simples. En efecto, la instrucción

```
LET A=10
```

incluye dos operaciones. Como ya vimos en las primeras lecciones: En primer lugar el BASIC comprueba si ya existe la variable A. Si no es así, la crea automáticamente y le asigna una porción de la memoria principal del ordenador. En segundo lugar se almacena el número 10 en A.

La creación y asignación automática de la memoria era posible puesto que sólo se reservaba espacio para un único dato. Cuando se trata de almacenar una lista (o tabla), el BASIC necesita saber cuántos elementos componen dicho conjunto, puesto que debe reservar espacio para cada uno. Esta es la razón por la cual hay que establecer las dimensiones de una lista o tabla antes de usarla. Además, en general, es necesario que cada dato contenga un valor diferente.

10.2.1 Instrucción DIM

Para realizar la operación de reserva de memoria se utiliza la instrucción DIM. La palabra DIM es una abreviatura de dimensión. Con esta instrucción se establece cuántas dimensiones tiene una variable y cuántos elementos tiene en cada dimensión. De esta forma el BASIC sabe cuánta memoria debe reservar para dicha variable.

La reserva de memoria se hace mediante la instrucción DIM

La forma general de esta instrucción consiste en escribir la palabra DIM y, a continuación, el nombre de la variable, la cual llevará entre paréntesis la dimensión o lista de dimensiones.

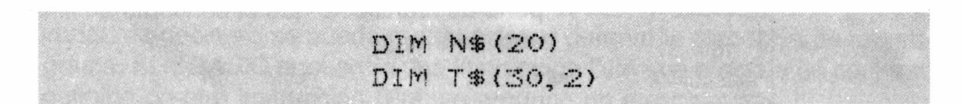
Por ejemplo:

```
DIM A(4)
DIM B(3,4)
DIM T(3,3,3)
```

En la figura 1 se muestra una representación visual de los tres ejemplos indicados de variables dimensionadas. En el primer ejemplo (Fig. 1a) la variable *A* será una lista de cuatro elementos. Por tanto se reservan cuatro celdas o casillas. En el segundo ejemplo (Fig. 1b), la variable *B* es una tabla con tres filas y cuatro columnas. Por tanto, el total de elementos es $3 \times 4 = 12$. Finalmente, en el tercer ejemplo (Fig. 1c), la variable *T* se define como un conjunto de tres dimensiones, cada una de las cuales tiene tres elementos. En total, por tanto, la variable *T* tiene $3 \times 3 \times 3 = 27$ elementos.

10.2.2 Variables textuales

En los tres primeros ejemplos del apartado anterior las variables eran de tipo numérico. Se pueden emplear también los conjuntos dimensionados para almacenar datos de tipo textual. Para ello utilizaremos el mismo procedimiento que utilizamos para las variables simples, es decir, colocaremos el símbolo dólar ($) al final del nombre de la variable. Por ejemplo:



```
DIM N$(20)
DIM T$(30,2)
```

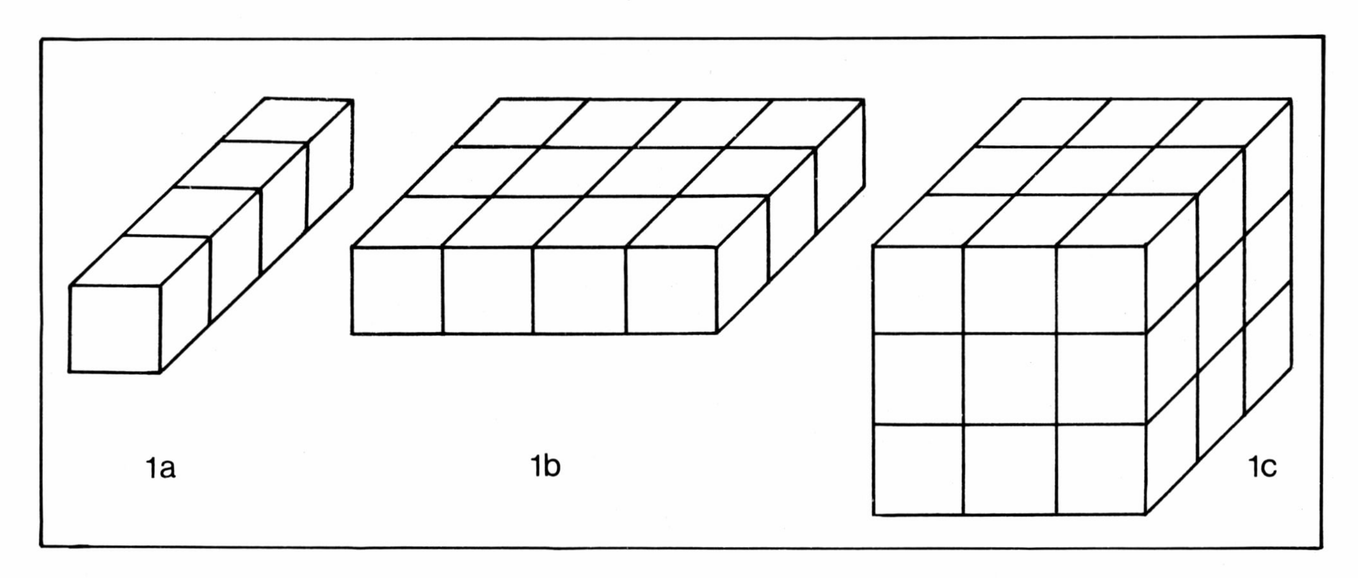


Figura 1. Representación visual de los conjuntos dimensionados. 1a) Conjunto de una dimensión o lista. 1b) Conjunto de dos dimensiones. 1c) Conjunto de tres dimensiones.

En el primer caso, la variable N$ es una lista de 20 textos. En el segundo caso, la variable T$ es una tabla de 30 filas y 2 columnas.

Todos los elementos de una variable dimensionada de tipo textual son a su vez textos. En general, cada uno de los elementos puede contener un número de letras (o símbolos) distinto de los demás elementos de la variable. Este número, en la mayoría de variantes del BASIC es inferor a 255.

10.2.3 Filas y columnas

Cuando se emplean tablas de datos (conjuntos de dos dimensiones) nos podríamos preguntar qué convenio utiliza el ordenador para las filas y las columnas. En otras palabras, la pregunta es si la primera dimensión equivale a las filas y la segunda a las columnas o viceversa. La respuesta es un poco sorprendente: al ordenador le es totalmente indiferente el convenio que utilicemos. Lo único que hay que tener en cuenta es mantener el mismo criterio a lo largo del programa. Cuando se empieza un programa que utiliza conjuntos dimensionados se elige un convenio, no importa cuál, sobre las filas y columnas (si el conjunto tiene más de dos dimensiones, el convenio debe abarcarlas todas). En el resto del programa debemos mantener el mismo criterio a fin de que los resultados sean coherentes.

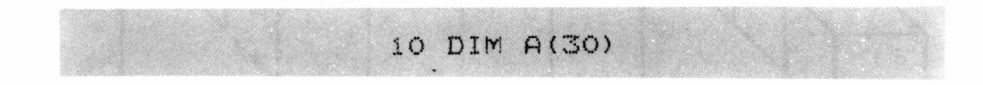
El nombre de fila o columna es una cuestión de convenio

Si esta aparente equivalencia entre filas y columnas nos parece difícil de entender, hay que recordar que el concepto de filas y columnas depende de la orientación con que miremos la tabla. Si nos imaginamos una tabla escrita sobre un papel, diremos que las líneas verticales son las columnas y las líneas horizontales son las filas. Sin embargo, si ahora giramos el papel 90 grados, las filas pasan a ser columnas y viceversa. A pesar de este cambio de orientación, los elementos son los mismos y, lo que es más importante, la situación relativa entre ellos no ha cambiado. Este hecho pone de manifiesto que el concepto de fila y columna tiene una importancia secundaria.



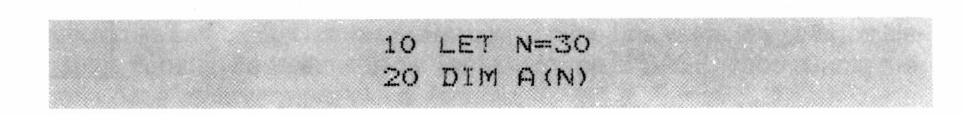
10.2.4 Extensiones de la instrucción DIM

En la mayor parte de las variantes del BASIC, la instrucción DIM tiene, además de las especificaciones vistas anteriormente, algunas ampliaciones. En primer lugar, para indicar las dimensiones se pueden emplear variables previamente definidas en lugar de números concretos. Por ejemplo, la instrucción



```
10 DIM A(30)
```

sería equivalente a



```
10 LET N=30
20 DIM A(N)
```

Como veremos más adelante, este procedimiento tiene sus ventajas para manejar listas o tablas de longitud determinada.

En una sola instrucción DIM se pueden declarar varias variables

En segundo lugar, en una sola instrucción DIM se pueden establecer las dimensiones de muchas variables. Así, por ejemplo, las dos instrucciones

```
10 DIM N$(10)
20 DIM T(25,2)
```

se podrían resumir en una sola, quedando

```
10 DIM N$(10),T(25,2)
```

10.2.5 Empleo de la instrucción DIM

Como ya hemos mencionado anteriormente, la instrucción DIM sirve para que el BASIC reserve memoria donde almacenar los conjuntos dimensionados. Por esta razón, la instrucción DIM debe usarse siempre *antes* de manejar cualquiera de los elementos del conjunto.

Aunque la instrucción DIM puede estar situada en cualquier lugar del programa (pero precediendo siempre a las instrucciones que utilizan los elementos del conjunto), se suelen colocar agrupadas al principio del programa. De este modo, el programador sabe enseguida qué conjuntos utiliza y qué dimensiones tienen.

Una vez establecidas las dimensiones de una lista o tabla no se pueden variar; es decir, no se puede redimensionar durante la ejecución de un programa. Si el BASIC encuentra una instrucción DIM que afecta a un conjunto definido en otra instrucción DIM, se produce un error.

No es posible cambiar las dimensiones una vez se ha definido

Conviene recalcar que la imposibilidad de cambiar de dimensión se refiere sólo durante la ejecución del programa. Antes de utilizar el comando RUN se pueden cambiar las dimensiones sin ningún problema. Lo que no está permitido es utilizar dos veces la instrucción DIM para la misma variable.

Finalmente hay que señalar que el tamaño máximo de las listas y tablas tiene un límite distinto según el modelo de ordenador. La instrucción DIM reserva espacio en la memoria principal del ordenador. La cantidad de memoria disponible nos indicará el tamaño de los conjuntos dimensionados que podemos utilizar.

10.3 UTILIZACION DE CONJUNTOS DIMENSIONADOS

Hasta ahora hemos visto cómo definir los conjuntos dimensionados. En este apartado veremos la utilización de los elementos de una lista o tabla dentro de las instrucciones normales.

Es necesario indicar la posición del elemento para acceder a un dato de un conjunto dimensionado

Para simplificar, nos referiremos por el momento a una lista que es un conjunto de una sola dimensión. Posteriormente lo ampliaremos para los conjuntos de más dimensiones.

Como es lógico no podemos referirnos a un dato contenido en una lista utilizando simplemente el nombre de la variable. Hay que indicar de alguna manera su posición dentro de la lista. Los elementos de una lista son correlativos de forma que podremos referirnos a ellos por su número de orden (tercero, séptimo, etc). Este número de orden se denomina *subíndice*. Este nombre proviene del Algebra matricial pues ya hemos mencionado que hay una relación muy estrecha entre los conceptos de lista y tabla por un lado y los conceptos de vector y matriz por otro.

Para referirnos a un elemento de una lista se escribe el nombre de la variable y a la derecha, entre paréntesis, el subíndice. Por ejemplo, supongamos que definimos la lista A con 4 elementos y queremos almacenar el varlor 85.7 en el segundo elemento de la lista. Escribiremos entonces

```
DIM A(4)
LET A(2)=85.7
```

La segunda instrucción nos muestra la forma general de cómo se utilizan los elementos de una lista. El contenido de la lista se observa en la figura 2. Todos los elementos son ceros, excepto el segundo donde hemos almacenado el dato 85.7. Si ahora tecleamos

```
PRINT A(1),A(2)
```

el primer resultado es un cero ya que A(1), es decir el primer elemento de la lista está vacío. El segundo resultado será el número 85.7 que corresponde a A(2). Los elementos de una lista se pueden emplear exactamente

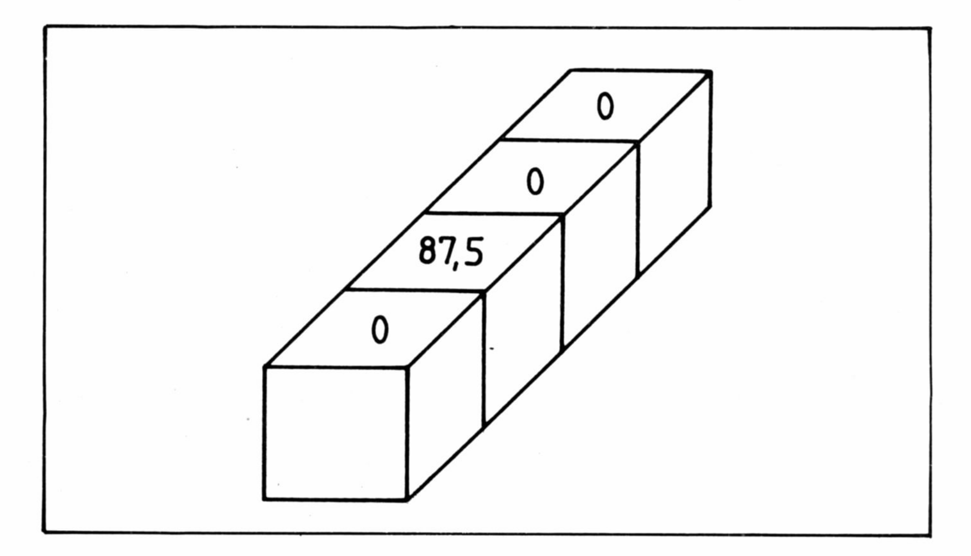


Figura 2. Lista numérica con su contenido.

en los mismos lugares donde se empleará una variable simple. Por tanto, si ahora escribimos

```
LET A(4)=A(2)
```

significa que realizamos una copia del valor de A(2) sobre A(4).

En este momento hay dos elementos (el segundo y el cuarto) de la lista A que tienen el mismo valor.

Hay que distinguir contenido y posición

Es muy importante entender la diferencia entre contenido y posición. En el ejemplo anterior, el número 2 y el 4 indican las posiciones de la lista y el número 87.5 es el contenido. En este ejemplo, la diferencia es muy evidente, pero a veces no lo es tanto. Si escribimos

```
LET A(2+1)=4
```

significa que en la posición o subíndice 3 (2 más 1) se almacena el número 4 (contenido). Obsérvese que el elemento A(2) no ha sufrido variación. La instrucción

```
PRINT A(3)
```

escribirá un 4 como resultado.

Todo lo anterior también es válido para las variables alfanuméricas o textuales. Por ejemplo:

```
DIM B$(4)
LET B$(2)="CASA"
```

Hemos definido una lista B$ que contiene 4 textos. En la segunda posición de la lista hemos almacenado la palabra «CASA».

10.3.1 Subíndices calculados

Como hemos visto en el ejemplo del apartado anterior, no hace falta emplear números concretos para indicar un subíndice, sino que se pueden emplear expresiones numéricas. El resultado de la expresión se toma como subíndice. Lógicamente, si el resultado es fraccionario, se suprimen los decimales puesto que no tiene sentido referirse por ejemplo al elemento 3.5.

La principal ventaja de permitir expresiones numéricas en el cálculo del subíndice está en que, como consecuencia, también está permitido el empleo de variables en este cálculo. Por ejemplo:

```
LET I=1
LET A(I)=20
```

El cálculo de subíndices se hace mediante expresiones

El valor 20 se almacenará en el elemento 1 de la lista puesto que I valía 1. Estudiando con atención este par de instrucciones, vemos que con sólo variar el valor de I, el número 20 se almacenará en cualquiera de las posiciones de la lista. Este hecho es fundamental, pues nos permitirá escribir programas con pocas instrucciones que, por el contrario, manejarán gran cantidad de datos.

Por esta razón, las instrucciones FOR/NEXT son compañeras inseparables de los conjuntos dimensionados. La variable de la instrucción FOR se emplea en el cálculo del subíndice.

10.3.2 Subíndices de una tabla

Cuando hay que referirse a un elemento de una tabla se indican tantos subíndices como dimensiones tiene la tabla. En general se emplean tablas de dos dimensiones y sólo muy raramente se emplean de tres o más dimensiones. Los diversos subíndices se sitúan dentro del paréntesis y se separan entre sí mediante comas (,).

Por ejemplo:

```
DIM T(4,4)
LET T(1,2)=87
```

En este caso se asigna el valor 87 al elemento de subíndices (1,2) de la tabla T.

10.3.3 Elemento cero

En el BASIC estándar, al definir un conjunto dimensionado se reserva espacio también para los elementos de subíndice cero. Por ejemplo, la instrucción

```
DIM X(4)
```

reserva espacio para los elementos X(0), X(1), X(2), X(3) y X(4).

Asimismo, la instrucción

```
DIM T(4,4)
```

reserva espacio para

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4
3,0	3,1	3,2	3,3	3,4
4,0	4,1	4,2	4,3	4,4

Sin embargo, no utilizaremos en nuestros ejemplos, excepto para casos muy concretos, los elementos con subíndice cero.

Al utilizar un elemento de un conjunto, el subíndice debe estar comprendido entre cero y el valor máximo de la dimensión. Si se usa un valor fuera de estos límites, se produce un error.

RESUMEN

Cuando hay que manipular muchos datos del mismo tipo, es necesario, disponer de listas o tablas de conjuntos dimensionados.

El BASIC dispone de unas variables que se refieren a un conjunto dimensionado completo.

Esta variable memoriza más de un dato elemental, es decir, número o texto; de aquí su nombre de conjunto.

Se le añade dimensionado, pues la disposición de los datos se asemeja a distintas disposiciones en el espacio.

Una lista es un conjunto seguido de datos. En lenguaje matemático se denomina también vector. Se trata del conjunto de una dimensión pues un elemento está lógicamente entre otros dos.

Una tabla es un conjunto de dos dimensiones, una dimensión es la que en un papel denominamos *filas* y la otra *columnas*. En términos matemáticos se denomina *matriz*.

De hecho es posible trabajar con más dimensiones, aunque el número de dimensiones más frecuente es uno o dos.

La principal característica de los conjuntos dimensionados es la homogeneidad, es decir, todos los datos son de las mismas características. No es posible mezclar números con textos y viceversa.

La definición en BASIC de los conjuntos dimensionados es más complicada que el de una variable sencilla.

Las normas referentes a los nombres de los conjuntos dimensionados son las mismas que las que se utilizan para las variables simples. El dólar al final del nombre indica que la tabla está construida con elementos textuales; en caso contrario, es que sus valores son numéricos.

Los nombres pueden tener alguna restricción respecto a los que tienen las variables simples.

En el momento de la definición, es necesario especificar la cantidad de elementos que contiene el conjunto, a fin de que el BASIC reserve la cantidad de memoria necesaria.

El BASIC dispone de la instrucción DIM para realizar la reserva de memoria. Es una abreviatura de dimensión.

La instrucción tiene dos misiones: establecer cuántas dimensiones tiene el conjunto y de cuántos elementos consta.

El formato general de la instrucción es:

DIM (nombre) (e1,e2,e3...)

en donde,

(nombre), define el *nombre* que utilizaremos para referirnos a este conjunto. Si el nombre termina con el símbolo dólar significa que el conjunto es de textos; en caso contrario es de números. El e1, e2, e3, son expresiones numéricas que indican el número de elementos que tiene cada dimensión. El número de expresiones que colocamos separados por comas es el número de dimensiones que tiene el conjunto dimensionado.

Además en una misma instrucción DIM se pueden definir varios conjuntos dimensionados, basta separar cada uno de ellos por una coma.

Es corriente dar el nombre de filas y columnas a cada una de las dimensiones del conjunto. Este nombre no está en principio asociado a la primera o segunda dimensión que hemos especificado, pues depende del uso que realicemos de cada una de ellas. Sólo es necesario ser coherente, después de tomar una decisión. Si a la primera dimensión se le llama fila, es necesario mantener esta nomenclatura a lo largo de todo el programa.

La instrucción DIM debe realizarse antes de utilizar los elementos que define. Además sólo debe realizarse una sola vez. Aunque se puede colocar en cualquier lugar del progrma se suelen agrupar al principio del mismo, para poder utilizar los elementos enseguida.

Para referirnos a un elemento concreto de un conjunto dimensionado se utiliza el subíndice, que nos da el número de orden del elemento dentro de una dimensión.

Para acceder a un elemento de un conjunto dimensionado se especifica el nombre del conjunto seguido del subíndice encerrado entre paréntesis.

En el interior del paréntesis se puede colocar una expresión en la que intervengan otras variables numéricas.

Cuando un conjunto contiene más de una dimensión se separan cada una de las expresiones que calcula cada uno de los subíndices mediante la coma.

El BASIC permite que el valor del subíndice sea cero. De hecho la instrucción DIM reserva espacio para el elemento cero.

Cuando definimos DIM A(3), reservamos de hecho 4 números, el que se asocia al subíndice 0, 1, 2 y 3.

EJERCICIOS AUTOCOMPROBACION

Complete las frases siguientes:

1. Las listas o tablas se utilizan para manipular datos del mismo ....

2. El BASIC dispone de unas variables para el almacén de muchos datos que se denominan .... dimensionados.

3. Las variables conjunto dimensionado memorizan más de un ....

4. Una lista es un conjunto dimensionado de .... dimensión. También se llama ....

5. Una tabla es un conjunto dimensionado de .... dimensiones. También se llama ....

6. El número de .... de un conjunto dimensionado puede ser grande, aunque lo más frecuente es uno o dos.

7. No es posible que un conjunto dimensionado contenga simultáneamente textos y ....

8. El nombre de los conjuntos dimensionados puede tener alguna .... según el tipo de BASIC.

9. El nombre de una variable conjunto dimensionado que termina en .... indica que sus elementos son textuales.

10. La instrucción DIM debe realizarse .... de utilizar los elementos que define.

11. El método para indicar un elemento del conjunto dimensionado es utilizar el número de orden relativo al .... del conjunto dimensionado.

12. El número de orden de un elemento dentro de un conjunto dimensionado se denomina ....

13. El subíndice de un elemento se encierra entre .... detrás del nombre del conjunto dimensionado.

14. Cuando un conjunto dimensionado tiene varias dimensiones los subíndices de cada una de ellas se separan por ....

15. Es posible utilizar en el Basic estándar el .... que vale cero.

Rodee con un círculo la V si la instrucción es verdadera y rodee la F si la instrucción es falsa.

16. DIM A(B$) V F

17. DIM A$(B) V F

18. DIM A(7,) V F

19. DIM A(8,3),B(2) V F

20. DIM J,B(J) V F

Sea la tabla sigúiente:

A(1)=2 : A(2) = 23 : A(3) = 32 : A(4) = 85

Encierre en un círculo la respuesta que corresponda a la alternativa correcta.

21. ¿Cuál es el subíndice cuyo elemento contiene un 23?

a) 1

b) 2

c) 3

d 4

22. ¿Qué valor se imprimirá con la instrucción?

PRINT A (A(1)+2)

a) 2
b) 23
c 32
d) 85

23. ¿Cuál es el subíndice del mayor elemento?

a) 1
b) 2
c) 3
d) 4

24. ¿Cuál es el elemento de mayor subíndice?

a) 2
b) 23
c) 32
d) 85

25. ¿Qué valor tiene el elemento de subíndice 3?

a) 85
b) 23
c) 32
d) 2



10.4 PROCEDIMIENTOS BASICOS

En este apartado veremos una serie de procedimientos y algoritmos básicos para manejar los conjuntos. Podemos considerar a estos procedimientos como las piezas fundamentales para construir programas más complejos.

10.4.1 Llenar un conjunto

Para almacenar información en un conjunto podemos emplear la instrucción LET de la forma que ya conocemos. Por ejemplo:

```
10 DIM A(4)
20 LET A(1)=60
30 LET A(2)=20
40 LET A(3)=47
50 LET A(4)=56
60 PRINT A(2)+A(4)
70 PRINT A(1)+A(3)
```

En la línea 10 definimos el conjunto A con cuatro elementos numéricos. En las líneas comprendidas entre la 20 y la 50 se almacenan los datos en el conjunto. Finalmente, en las líneas 60 y 70 se escriben respectivamente las sumas de los elementos pares (2 y 4) y de los impares (1 y 3). Estas dos últimas líneas no tienen importancia. Simplemente están para comprobar que hemos utilizado correctamente las instrucciones anteriores para entrar datos.

Si los datos para llenar el conjunto hay que entrarlos por el teclado en el momento de la ejecución se emplea entonces la instrucción INPUT en lugar de LET. El programa anterior quedaría:

```
10 DIM A(4)
20 INPUT A(1)
30 INPUT A(2)
40 INPUT A(3)
50 INPUT A(4)
60 PRINT A(2)+A(4)
70 PRINT A(1)+A(3)
```

La utilización de variables en los subíndices permite facilitar el trabajo de programación

Observamos que las líneas 20, 30, 40 y 50 son muy parecidas entre sí y únicamente se diferencian en el subíndice de la variable. Es el momento de aprovechar las ventajas de las instrucciones FOR/NEXT. El programa anterior lo cambiaremos por:

```
10 LET N=4 : DIM A(N)
20 FOR I=1 TO N
30     INPUT A(I)
40     NEXT I
50 PRINT A(2)+A(4)
60 PRINT A(1)+A(3)
```

Aquí hemos introducido varias modificaciones. En primer lugar, en la línea 10 empleamos la variable N para establecer las dimensiones de A. Esto tiene la ventaja de que memorizamos el número de elementos que tiene el conjunto. De esta forma podemos utilizar la variable N en la línea 20. En esta línea establecemos un bucle que se repetirá desde 1 hasta N. La línea 30 es importante. En ella utilizamos la variable I de la instrucción FOR para determinar el subíndice. De esta forma sólo es necesario emplear una instrucción INPUT. La línea 40 es la que cierra el bucle. Las líneas 50 y 60 son iguales a las líneas 60 y 70 del programa anterior.

Flexibilidad del programa

En este ejemplo hay que remarcar dos cosas. En primer lugar el ahorro de instrucciones. Esta forma de escribir el programa es mucho más simple que teclear todas las instrucciones INPUT. En segundo lugar el programa es muhco más flexible. Si en lugar de emplear un conjunto de 4 elementos, deseamos utilizar uno de 20 elementos, únicamente hay que realizar un cambio en la línea 10. En esta línea sustituiremos el valor 4 que se asigna a N por el valor 20. La instrucción queda:

```
10 LET N=20 : DIM A(N)
```

Tanto la instrucción DIM como la instrucción FOR de la línea 20 no sufren variación, ya que la variable N contiene siempre el límite real del conjunto A.

Al efectuar un RUN y probar el programa, éste nos pedirá que entremos 4 números (o 20 según el valor de N). Notaremos que una vez entrados unos cuantos, se hace difícil recordar cuál es la posición del elemento que estamos tecleando. Para facilitar las cosas añadiremos una instrucción que nos informe de la posición para la cual entramos el dato. Esta instrucción será:

```
25 PRINT "Elemento ";I
```

De esta forma, antes de pedirnos el dato, aparecerá en pantalla un mensaje del tipo

Elemento 1

El número del final irá variando según el valor de I.

Todo lo que acabamos de ver para una lista numérica también se aplica para las variables de tipo textual. Por ejemplo:

```
10 DIM B$(4)
20 LET B$(1)="CASA "
30 LET B$(2)="ARBOL "
40 LET B$(3)="GRANDE"
50 LET B$(4)="ALTO"
60 PRINT B$(1)+B$(3)
70 PRINT B$(2)+B$(4)
```

Este programa es idéntico al primer ejemplo del conjunto numérico A. Las instrucciones 60 y 70 escribirán en pantalla

CASA GRANDE

ARBOL ALTO

De igual manera, se emplea la instrucción INPUT cuando los datos se teclean en el momento de ejecución. El programa quedará:

```
10 LET N=4 : DIM B$(N)
20 FOR I=1 TO N
30    PRINT "ELEMENTO ";I
40    INPUT B$(I)
50    NEXT I
60 PRINT B$(1)+B$(3)
70 PRINT B$(2)+B$(4)
```

Cuando se trata de introducir datos en una tabla utilizaremos un procedimiento parecido pero con dos bucles. Puesto que llenaremos filas y columnas, colocaremos dos bucles. Por cada fila, leeremos todas las columnas. Por tanto dispondremos uno de los bucles en el interior del otro. Conviene recordar lo que mencionamos acerca de la equivalencia entre filas y columnas. Antes de diseñar un programa hay que establecer un convenio. Por ejemplo, el primer subíndice serán las filas y el segundo serán las columnas.

Supongamos que tenemos la tabla de la figura 3. Las filas son las asignaturas y las columnas son dos exámenes distintos. El siguiente programa permitirá leer esta tabla.

```
10 DIM C(3,2)
20 FOR I=1 TO 3
30    FOR J=1 TO 2
40       INPUT C(I,J)
50       NEXT J
60    NEXT I
```

Cada dimensión requiere su bucle o instrucción FOR/NEXT

El bucle exterior determina el número de fila (o asignatura) y el bucle interior determina el número de columna (o examen). Las notas deberán teclearse en el orden siguiente: 7, 8, 4, 6, 6 y 5. En este momento la tabla

	Exam. 1	**Exam. 2**
Asignatura 1	7	8
Asignatura 2	4	8
Asignatura 3	6	5

Figura 3. Tabla con notas de tres asignaturas correspondientes a dos exámenes.

C contiene las notas en el orden adecuado. Lo comprobaremos escribiendo la instrucción

```
PRINT C(2,1)
```

que imprimirá el número 4 que corresponde a la nota de la asignatura 2 en el examen 1.

Si deseáramos entrar las notas en orden inverso, es decir, primero todas las notas de un examen, colocaríamos los bucles de la siguiente manera:

```
10 DIM C(3,2)
20 FOR J=1 TO 2
30    FOR I=1 TO 3
40       INPUT C(I,J)
50       NEXT I
60    NEXT J
```

En este caso, el orden en que deberían teclearse las notas sería: 7, 4, 6, 8, 6 y 5. Si no hemos cometido ningún error al entrar los datos, la tabla C contiene exactamente los mismos datos que antes. Si efectuamos de nuevo la instrucción

```
PRINT C(2,1)
```

el resultado sigue siendo 4.

Una idea importante que nos enseña este ejemplo es que una tabla se maneja siempre con dos bucles situados uno en el interior del otro («anidados»).

También sería interesante aplicar aquí el procedimiento de guardar el tamaño del conjunto en una variable. Como es lógico, por tratarse de una tabla necesitaremos dos variables. El programa quedaría entonces

```
10 LET N=3 : LET M=2
20 DIM C(N,M)
30 FOR J=1 TO M
40    FOR I=1 TO N
50       INPUT C(I,J)
60       NEXT I
70    NEXT J
```

Las instrucciones comprendidas entre la 30 y la 70 nos sirven para cualquier tabla, sea cual sea su tamaño.

10.4.2 Escribir un conjunto

Si sólo hay que imprimir el contenido de algunos elementos del conjunto emplearemos la instrucción PRINT de forma normal. Si por el contrario, es necesario escribir todo el conjunto o una parte importante del mismo, aprovecharemos las ventajas que nos facilitan las instrucciones FOR/NEXT.

Volvamos ahora al ejemplo donde utilizábamos la lista A. Supongamos que queremos escribir todos los elementos entrados en pantalla. El programa quedará:

```
10 LET N=4 : DIM A(N)
20 FOR I=1 TO N
30     PRINT "ELEMENTO ";I
40     INPUT A(I)
50     NEXT I
60 FOR I=1 TO N
70     PRINT I,A(I)
80     NEXT I
```

Las líneas situadas entre la 10 y la 50 sirven para entrar los datos. La línea 60 establece el bucle de escritura que, como de costumbre, irá de 1 hasta N. En la línea 70 hacemos escribir el valor del subíndice y el contenido del subíndice. De esta forma el propio programa nos escribe numerados los elementos facilitando su lectura. Finalmente la línea 80 cierra el lazo de la línea 60.

Estas tres últimas líneas escribirán todo el conjunto en el mismo orden en que se han entrado los datos.

Ejecute ahora el programa escribiendo RUN y déle un número a cada elemento, según le pide el programa.

Puede darse el caso de que no se dese escribir toda la lista. Por ejemplo, supongamos que sólo queremos escribir los elementos cuya posición sea un número impar (elemento 1, elemento 3, etc). Entonces las tres últimas líneas quedarían:

```
60 FOR I=1 TO N STEP 2
70     PRINT I,A(I)
80     NEXT I
```

Añadiendo la potencia del FOR/NEXT a las tablas de datos se pueden realizar

Hemos añadido STEP 2 a la línea 60, de modo que el bucle avanzará de dos en dos, empezando desde 1.

Si queremos escribir la lista en orden inverso al entrado, cambiaremos de nuevo la línea 60, quedando:

```
60 FOR I=N TO 1 STEP -1
70     PRINT I,A(I)
80     NEXT I
```

Con estas instrucciones, la lista se escribirá desde N hasta 1. Como podemos comprobar, las instrucciones FOR/NEXT nos ofrecen una gran flexibilidad en el manejo de conjuntos.

Para escribir una tabla utilizaremos dos bucles. En el ejemplo de la tabla de notas, el programa quedará:

```
 10 LET N=3 : LET M=2
 20 DIM C(N, M)
 30 FOR I=1 TO N
 40     FOR J=1 TO M
 50         INPUT C(I, J)
 60         NEXT J
 70     NEXT I
 80 FOR I=1 TO N
 90     FOR J=1 TO M
100         PRINT C(I, J)
110         NEXT J
120     NEXT I
```

Las líneas comprendidas entre la 10 y la 70 son las que sirven para llenar la tabla C. A partir de la 80 y hasta la 120 establecemos los dos bucles de escritura. Observamos que la estructura de los bucles es idéntica a la de lectura y sólo se cambia la instrucción INPUT por la PRINT.

Ocurre que, al ejecutar este programa, las notas se escriben consecutivamente en vertical. Esto es lógico puesto que la instrucción PRINT de la línea 100 salta de línea cada vez que se ejecuta. Si colocamos un punto y coma (;) al final de la instrucción, es decir:

```
100 PRINT C(I, J);
```

impediremos el salto de línea. Ahora las notas se escriben en horizontal, puesto que no se produce ningún salto de línea. Para que las notas queden en forma de tabla, hay que provocar un salto de línea cuando complete una fila. Una fila se completa cuando la variable J ha llegado al límite (a M). Entonces colocaremos una instrucción PRINT detrás de la línea 110. El programa quedará, a partir de la línea 80

```
 80 FOR I=1 TO N
 90     FOR J=1 TO M
100         PRINT C(I, J);
110         NEXT J
115     PRINT
120     NEXT I
```

Los resultados quedarán escritos en forma de tabla. No hay ninguna dificultad en girar 90 grados la tabla y escribir las asignaturas en horizontal y los exámenes en vertical (recordemos la equivalencia entre columnas y filas). Para realizar este giro, intercambiaremos el orden de los bucles

```
 80 FOR J=1 TO M
 90     FOR I=1 TO N
100         PRINT C(I,J);
110         NEXT I
115     PRINT
120     NEXT J
```

Las notas se escribirán como una tabla de tres columnas (o asignaturas) y dos filas (o exámenes). Probablemente este ejemplo nos habrá ayudado a comprender con mayor claridad todo lo que llevamos dicho sobre equivalencia entre filas o columnas. Observamos también que se respeta siempre el convenio inicial. Los elementos de la tabla se escriben siempre de la forma C(I,J). Este orden no se puede cambiar puesto que hemos establecido que el primer subíndice va de 1 a 3 y el segundo va de 1 a 2. Por tanto existe el elemento C(3,1) pero, en cambio, no existe el elemento C(1,3).

10.4.3 Máximo y mínimo de una lista

Una necesidad que surge frecuentemente en programación es localizar el valor máximo (o el mínimo) de una lista. Para localizarlo rastrearemos toda la lista. El proceso es el siguiente: En primer lugar se supone que el máximo es el valor inicial de la lista. A continuación se compara este supuesto máximo con el siguiente elemento. Si éste es mayor, se coloca en el lugar del máximo. En caso contrario no se realiza ninguna acción. Este proceso se repite para todos los elementos de la lista. El programa es el siguiente:

```
 10 LET N=5 : DIM A(N)
 20 FOR I=1 TO N
 30     PRINT I
 40     INPUT A(I)
 50     NEXT I
 60 LET M=A(1)
 70 FOR I=2 TO N
 80     IF A(I)>M THEN LET M=A(I)
 90     NEXT I
100 PRINT "MAXIMO=";M
```

En la línea 10 establecemos la lista A con N elementos (en este caso 5). Las líneas situadas entre la 20 y la 50 constituyen el ya conocido bucle

de entrada de datos. En este ejemplo le entraremos cinco datos numéricos cualesquiera. En la línea 60 empieza el proceso de localización del máximo, que lo almacenaremos en la variable M. En primer lugar suponemos que el máximo es el primer elemento de la lista. Esta suposición es completamente arbitraria y se realiza únicamente para empezar con un valor. No hay ningún problema en asignar a la variable simple M un elemento de una lista, A(1). Ya dijimos que un elemento de un conjunto dimensionado es totalmente equivalente a una variable simple.

Rastreo y selección de un elemento de una lista

A continuación se encuentra el bucle de rastreo. Este lazo va desde el segundo elemento hasta N. No hace falta empezar desde 1 puesto que el valor del primero ya está almacenado en M. En la línea 80 comprobamos si el elemento que estamos mirando es mayor que M. Si es así, sustituimos el valor antiguo de M por el nuevo máximo. La línea 90 cierra el bucle.

Una vez rastreada toda la lista, la variable M contendrá forzosamente el valor máximo. Finalmente, en la línea 100 el programa escribe su valor.

Si en lugar de localizar el valor máximo, se desea localizar el valor mínimo, el procedimiento es análogo. Unicamente hay que cambiar la pregunta. En el programa anterior, la línea 80 queda:

```
80 IF A(I)<M THEN LET M=A(I)
```

Ahora la variable M contiene el valor mínimo. Fijémonos que en la línea 60, la suposición inicial es que A(1) es el mínimo. Por supuesto también hay que cambiar el mensaje de la línea 100, que quedará:

```
100 PRINT "MINIMO=";M
```

Al probar este programa observamos que el resultado es únicamente el valor del máximo (o del mínimo) pero no nos indica en qué posición de la lista se encuentra dicho valor.

Para conocer esta posición utilizaremos una variable auxiliar donde la almacenaremos. El contenido de esta variable se actualizará cada vez que se modifique el valor de M, es decir, en las líneas 60 y 80. El programa anterior quedará de la siguiente manera a partir de la línea 60:

```
60 LET M=A(1) : LET P=1
70 FOR I=2 TO N
80     IF A(I)>M THEN LET M=A(I) : LET P=I
90     NEXT I
100 PRINT "MAXIMO=";M
110 PRINT "POSICION=";P
```

Muchas veces es conveniente memorizar la posición de un dato

La variable P es la que contiene la posición. Empieza con el valor 1 que corresponde a la posición de A(1). En la línea 80, si se almacena A(I) en M, entonces también se memoriza la posición I en la variable P. Finalmente en la línea 110 se escribe el valor de P.

Si en un programa se necesita localizar simultáneamente el valor máximo y el mínimo, no hace falta colocar dos bucles. Un solo bucle nos servirá para rastrear simultáneamente el máximo y el mínimo. En el siguiente programa veremos cómo:

```
 10 LET N=5 : DIM A(N)
 20 FOR I=1 TO N
 30     PRINT I
 40     INPUT A(I)
 50     NEXT I
 60 LET MA=A(1) : LET P1=1
 70 LET MI=A(1) : LET P2=1
 80 FOR I=2 TO N
 90     IF A(I)>MA THEN LET MA=A(I) : LET P1=I
110     IF A(I)<MI THEN LET MI=A(I) : LET P2=I
120     NEXT I
130 PRINT MA,P1
140 PRINT MI,P2
```

En este programa, el máximo se almacenará en MA y su posición en P1. El mínimo se almacenará en MI y su posición en P2. Dentro del lazo formado por las líneas 80 y 110, se incluyen las dos preguntas para determinar simultáneamente el valor del máximo y del mínimo.

Por supuesto, en todos estos ejemplos para el ordenador no constituye ningún problema si en lugar de tomar 5 elementos (línea 10), escogemos un número mucho mayor. Sin embargo, para el operador humano constituye una tarea muy fatigosa, introducir centenares o incluso miles de datos.

10.4.4 Suma de los elementos

Otro de los algoritmos básicos del manejo de conjuntos es la suma de los elementos de una lista o de una tabla. El procedimiento para una lista es el siguiente:

```
 10 LET N=5 : DIM A(N)
 20 FOR I=1 TO N
 30     PRINT I
 40     INPUT A(I)
 50     NEXT I
 60 LET S=0
 70 FOR I=1 TO N
 80     LET S=S+A(I)
 90     NEXT I
110 PRINT "SUMA=";S
```

Variables que acumulan resultados resumen al manejar las tablas

Las líneas comprendidas entre la 10 y la 50 sirven para definir y llenar la lista A como ya conocemos de los ejemplos anteriores. En la línea 60 definimos la variable S que servirá para acumular el resultado de las sumas. Su valor inicial (antes de sumar nada) es cero. A continuación se establece un bucle formado por las líneas 70 y 90. Este bucle se realiza para todos los elementos de la lista cuyo valor se acumula en S (línea 80). En la línea 100, fuera ya del bucle, se escribe el valor total de la suma de elementos de la lista. En resumen, la estructura básica de este sistema se basa en inicializar con un cero la variable que acumulará el resultado y a continuación el proceso de suma sobre esta variable repetido para todos los elementos.

Si hay que sumar los elementos de una tabla, se aplica el mismo procedimiento básico del caso anterior. Pero ahora tenemos tres posibilidades: suma de columnas, suma de filas y suma total. Para ilustrar los tres casos, volveremos al ejemplo de la tabla de notas (Fig. 3). Aunque pueda parecer lo contrario, el caso más sencillo de los tres es obtener la suma global de toda la tabla. El programa es el siguiente:

```
 10 LET N=3 : LET M=2
 20 DIM C(N,M)
 30 FOR I=1 TO N
 40    FOR J=1 TO M
 50       INPUT C(I,J)
 60       NEXT J
 70    NEXT I
 80 LET S=0
 90 FOR I=1 TO N
100    FOR J=1 TO M
110       LET S=S+C(I,J)
120       NEXT J
130    NEXT I
140 PRINT "SUMA=";S
```

Las líneas comprendidas entre la 10 y la 70 sirven para entrar los datos por el teclado. Las notas deberán entrarse en el orden: 7, 8, 4, 6, 6 y 5. A partir de la línea 80 empieza el proceso de suma. Como podemos observar, el procedimiento es análogo al del caso anterior. La única diferencia es que el bucle es doble puesto que hay que operar con dos subíndices. En la línea 140 se escribe la variable S que contendrá la suma global de todos los elementos de la tabla.

Para sumar una columna de una tabla hay que definir primero qué columna deseamos sumar. Supondremos, por convenio, que el primer subíndice son filas y el segundo son columnas. El programa anterior lo modificaremos a partir de la línea 80, puesto que la parte inicial queda invariable. La parte que realiza la suma queda de la siguiente forma:

```
 80 LET S=0
 90 LET V=1
100 FOR I=1 TO N
110     LET S=S+C(I,V)
120     NEXT I
130 PRINT "COLUMNA=";V;"SUMA=";S
```

La variable V definida en la línea 90 es la que establece el número de columna. En este caso se le hace sumar la columna 1. Si se coloca un 2 en V se efectuaría la suma de la segunda columna. Cualquier otro número daría lugar a un error ya que sólo existen 2 columnas. En la línea 110 se acumula en S el valor de la suma de los elementos cuyo subíndice de columna es V. Finalmente, en la línea 130 se escribe el número de columna y su suma. El resultado de la primera columna es 17 y el de la segunda es 19.

La acumulación de resultados en una fila o columna, en las tablas

Para sumar filas se intercambia el papel que juega cada subíndice. El segmento del programa anterior queda:

```
 80 LET S=0
 90 LET F=1
100 FOR J=1 TO M
110     LET S=S+C(F,J)
120     NEXT J
130 PRINT "FILA=";F;"SUMA=";S
```

Ahora la variable F establece el valor de la fila. Los valores de F válidos son 1, 2 y 3. Comparemos los subíndices de C en la línea 110 y veremos que ahora el primer subíndice (F) permanece fijo, mientras que el segundo (J) varía de 1 a M. Las sumas de las tres filas son, respectivamente 15, 10 y 11.

A veces hay que calcular la suma para todas las columnas (o filas) de una tabla. No es una buena solución programar un bucle para cada una de las columnas ya que, según el tamaño de la tabla, el programa podría ser enorme. En su lugar, se define una lista S de sumas con tantos elementos como columnas (o filas) tenga la tabla. Entonces el proceso de suma se sitúa dentro de un bucle que se repite para todos los elementos de S.

Aprovecharemos también aquí la parte inicial del programa con una excepción. Añadiremos la línea 25

```
25 DIM S(M)
```

La suma de filas y columnas se hace con un conjunto dimensionado de una dimensión

Esta línea nos define la lista S que contendrá las sumas de las M columnas. Cuando se ejecuta una instrucción DIM, todos los elementos de la variable definida en ella se ponen a cero (si la variable es numérica, claro está). Por tanto, ahora no será necesario colocar un cero inicialmente a

todos los elementos de S antes de empezar a sumar. A partir de la línea 80, el programa queda:

```
 80 FOR J=1 TO M .
 90    FOR I=1 TO N
100       LET S(J)=S(J)+C(I,J)
110       NEXT I
120    NEXT J
130 FOR J=1 TO M
140    PRINT S(J)
150    NEXT J
```

Las líneas 90, 100 y 110 constituyen el lazo de suma de una columna. El número de la columna viene definido por J, cuyo valor lo establece el bucle mayor formado por las líneas 80 y 120. Remarcamos el hecho de que en la línea 100 los valores de los elementos de la columna J se almacenan en el elemento J de S. Finalmente, a partir de la línea 130 se escriben los resultados almacenados en S que en este caso son 2. El primero vale 17 y el segundo 19. Como es fácil comprobar, las instrucciones FOR/NEXT son compañeros inseparables de los conjuntos dimensionados.



10.4.5 Localización por número

Una operación frecuente es la localización de un elemento dentro de una lista. Esta búsqueda puede efectuarse de dos maneras, por número y por nombre. El primer tipo de búsqueda responde a la pregunta de «¿Qué es lo que hay en la posición número tal?». El segundo tipo responde a «¿En qué posición se encuentra el nombre tal?».

La localización por número es la más sencilla. Aunque se pueden realizar búsquedas en listas numéricas o textuales indistintamente, suelen ser de mayor utilidad las que se realizan sobre listas de textos. Por esta razón, emplearemos este tipo de variables en nuestro ejemplo.

```
10 REM Localizacion por numero
20 LET N=5 : DIM A$(N)
30 FOR I=1 TO N
40    INPUT A$(I)
50    NEXT I
60 INPUT "POSICION:";P
70 IF P=0 THEN STOP
80 PRINT A$(P)
90 GOTO 60
```

La búsqueda por número es directa. No se precisa rastreo

La línea 10 es un comentario informativo sobre el programa. A continuación (línea 20) se define la tabla alfanumérica o textual A$ de cinco elementos. Las líneas situadas entre la 30 y la 50 sirven para introducir los datos en A$ desde el teclado. Supongamos que le entramos la siguiente lista de nombres: MARIA, JUAN, ANA, JOSE y PEDRO. Seguidamente el programa nos pide la posición. Le responderemos con un número comprendido entre 1 y 5. Si le respondemos un cero, el programa lo toma como indicación de que queremos terminar la ejecución del programa. Esto se realiza en la línea 70, donde si se cumple que P vale cero, entonces se detiene el programa (instrucción STOP). Si respondemos un 4 por ejemplo, entonces el programa sigue y en la línea 80 nos escribirá el valor del cuarto elemento de A$ que es JOSE. El control se transfiere de nuevo a la línea 60 donde nos pide una nueva posición. Mientras le respondamos valores de la posición comprendidos entre 1 y 5, el proceso se repetirá indefinidamente. Para detener el programa, le entraremos un cero, como ya hemos mencionado.

Este programa serviría igualmente fuese cual fuese el tamaño de la lista. Es importante señalar que el ordenador tardará exactamente lo mismo en localizar cualquier elemento tanto si está situado al principio como al final de la lista.

10.4.6 Localización por nombre

La localización por nombre requiere el rastreo de la lista, y una salida del bucle si se encuentra

La localización por nombre es un poco más difícil de programar que la búsqueda por número. A diferencia del caso anterior, ahora puede ocurrir que el programa no localice el nombre buscado porque éste no se encuentre realmente en la lista. Por tanto, ahora hay que preveer dos casos, que haya éxito en la búsqueda o que no lo haya. El programa es el siguiente:

```
 10 REM Localizacion por nombre
 20 LET N=5 : DIM A$(N)
 30 FOR I=1 TO N
 40    INPUT A$(I)
 50    NEXT I
 60 INPUT "NOMBRE:";X$
 70 IF X$="" THEN STOP
 80 FOR I=1 TO N
 90    IF X$=A$(I) THEN GOTO 130
100    NEXT I
110 PRINT "NO ESTA"
120 GOTO 60
130 PRINT "POSICION=";I
140 GOTO 60
```

La primera parte, hasta la línea 50 es idéntica al ejemplo anterior excepto, claro está, el mensaje explicativo de la línea 10. Para probar el programa podemos emplear la misma lista de nombres.

En la línea 60 el programa pide el nombre a localizar, que se almacenará en X$. Podemos contestar cualquier nombre. Si respondemos con un texto vacío, representado por dos símbolos de comillas juntos (""), el programa se detiene como se ve en la línea 70. En caso contrario, el programa inicia la fase de búsqueda.

En la línea 80 empieza un proceso de rastreo donde se va preguntando si el elemento es igual a X$ (línea 90). Si no conciden, se incrementa el valor de I y se sigue el bucle. Supongamos que el programa llega al final del bucle y ningún elemento de A$ es igual a X$. En este caso, la ejecución prosigue por la línea 110 donde el programa nos informa del fracaso en la búsqueda. A continuación, pasa el control de nuevo a la línea 60.

Retrocedamos ahora a la línea 90. Si en algún momento durante el rastreo se encuentra un elemento de A$ idéntico a X$, entonces el programa abandona el bucle y continúa en la línea 130, donde escribe el valor de I. Notemos que este valor de I es el que corresponde al subíndice de A$ cuando hemos abandonado el bucle. Por tanto, I contiene la posición donde hemos localizado X$.

Introduzcamos en primer lugar los mismos nombres que antes al ejecutar el programa. Si en la línea 60 contestamos JUAN, el programa nos imprime un 2. Si escribimos PEDRO, nos escribe un 5, etc. Al escribir los nombres hay que recordar lo que ya se ha comentado respecto a la diferencia entre letras mayúsculas y minúsculas. No estará de más repasar en este punto el capítulo dedicado a comparación lexicográfica.

En la localización por nombre, el tiempo de búsqueda depende de la posición. Puesto que se realiza un rastreo, los elementos situados hacia el final de la lista tardan más en ser localizados. En nuestro ejemplo, al tratarse de una lista corta, no hay diferencia apreciable en el tiempo que tarda en localizar un elemento u otro. Sin embargo, en listas más largas, la diferencia puede ser significativa.

10.4.7 Fechas en letras

El objetivo de este programa es convertir una fecha escrita en números a la misma fecha escrita en letras. Por ejemplo, la fecha 21-10-84 se transforma en 21 de octubre de 1984. Este programa es bastante interesante puesto que incluye la aplicación de varias funciones, además del empleo de conjuntos dimensionados.

Para que el programa opere correctamente, la fecha debe tener la forma general DD-MM-AA, es decir, dos cifras para el día, mes y año. Entre los tres grupos de cifras se incluirá un símbolo de separación que puede ser un guión (-), una barra (/) o cualquier otro. El listado del programa es el siguiente:

```
10 REM Fecha en letras
20 DIM N$(12)
30 LET N$(1)="Enero"
40 LET N$(2)="Febrero"
50 LET N$(3)="Marzo"
60 LET N$(4)="Abril"
```

```
 70 LET N$(5)="Mayo"
 80 LET N$(6)="Junio"
 90 LET N$(7)="Julio"
100 LET N$(8)="Agosto"
110 LET N$(9)="Setiembre"
120 LET N$(10)="Octubre"
130 LET N$(11)="Noviembre"
140 LET N$(12)="Diciembre"
150 INPUT "FECHA (DD-MM-AA):";F$
160 IF F$="" THEN STOP
170 LET D$=LEFT$(F$,2)
180 LET M$=MID$(F$,4,2)
190 LET A$=RIGHT$(F$,2)
200 LET I=VAL(M$)
210 PRINT D$;" de ";N$(I);" del 19";A$
220 GOTO 150
```

La línea 10 es un comentario sobre el objetivo del programa. En la línea 20 se define la variable N$ de 12 elementos que contendrá la lista de los nombres de los meses. Las líneas situadas entre la 30 y la 140 sirven para almacenar dichos nombres. Se emplea la instrucción LET en lugar de INPUT porque entonces el operador se vería obligado a teclear la lista completa de los nombres cada vez que efectuara un RUN. En capítulos posteriores veremos algunos sistemas más eficientes para llenar un conjunto dimensionado.

En la línea 150 el programa pide la fecha a convertir. Supongamos que le contestamos «21-10-86». Este valor se almacena en F$. Notemos que la fecha hay que almacenarla en una variable textual pues, aunque contiene cifras, no constituye ciertamente un número.

La línea 160 se utiliza para que el operador pueda detener el programa. En efecto, si en lugar dè la fecha contestamos el texto vacío ("") entonces el programa se detiene.

El programa descompone ahora la fecha en sus tres elementos básicos (día, mes, año). Esta operación la realizan las instrucciones 170, 180 y 190. Después de efectuar estas instrucciones, la variable D$ contendrá, siguiendo con el ejemplo, el valor «21». Asimismo, las variables M$ y A$ contendrán los valores «10» y «86» respectivamente. Para fragmentar la variable F$, se han empleado las funciones textuales que se estudiaron en los primeros capítulos. En caso de existir alguna duda convendría realizar un repaso. Notemos que los tres datos (día, mes, año) siguen siendo de tipo textual pues provienen de la fragmentación de un texto (F$).

Los nombres de los meses son una aplicación de la localización por número

Otro de los puntos clave de este programa es convertir el valor del mes en su nombre. Es decir, pasar de «10» a octubre. Los nombres de los meses están almacenados en la lista N$. Por tanto si escribimos

```
PRINT N$(10)
```

el resultado será «Octubre». Como vemos, se trata del mismo procedimiento que el descrito en la localización por número. La solución parece inmediata. Sustituir el subíndice 10 por la variable que contiene el mes, de modo que según sea el valor de ésta, se escribirá el valor del mes asociado. Sin embargo, hay un problema. El valor del mes está almacenado como dato textual y no puede emplearse por tanto como subíndice de un conjunto. Ahora bien, existe una función que permite realizar la conversión de tipo textual a numérico. Esta función es VAL. En la línea 200 la variable I recibe el valor numérico de M$, en nuestro ejemplo, un 10.

Tenemos ya todas las piezas y sólo falta escribirlas. En la línea 210 se escriben D$ y A$ sin cambio alguno pues tanto el día como el año se escriben en cifras. Para escribir el mes, se utiliza N$(I). Para completar la frase se usan los textos fijos «de» y «del 19», de forma que el resultado sea «21 de Octubre del 1984». Finalmente, se pasa control de nuevo a la línea 150 para realizar la conversión de una nueva fecha.

Este programa funciona para cualquier fecha correcta. Sin embargo no realiza ninguna comprobación sobre su validez. Si se le entra una fecha incorrecta o que no se ajusta al formato DD-MM-AA, el programa escribirá resultados erróneos.

Si el valor del día es inferior a 10, por ejemplo 01-01-84, entonces la fecha convertida queda «01 de Enero del 1984». El cero inicial de «01» permanece puesto que se considera un dato de tipo textual. Para suprimirlo, lo pasaremos a numérico ya que entonces quedan eliminados los ceros de la izquierda por no ser significativos. La línea 210 quedará:

```
210 PRINT VAL(D$);" de ";N$(I);" del 19";A$
```

Al igual que en la línea 200, hemos empleado la función VAL para realizar la conversión. Notemos que este cambio no afecta a los días cuyo valor sea 10 o más. En cambio, la fecha «01-01-84» quedará escrita como «1 de Enero de 1984».

10.4.8 Cálculo de la media

Una operación que suele ir asociada a los conjuntos numéricos es el cálculo de la media aritmética. Recordemos que la media aritmética se define como la suma de los elementos dividido por el número de elementos. Así por ejemplo, si tenemos tres notas correspondientes a tres exámenes, cuyos valores son 4, 6 y 8, la nota media se calculará haciendo $4 + 6 + 8 = 18$ que dividido por 3 se obtiene $18 / 3 = 6$.

De acuerdo con la definición anterior está claro que la primera operación a realizar es la suma de los elementos. En uno de los procedimientos básicos vistos anteriormente ya hemos resuelto este problema. El cálculo de la media será, simplemente, una ampliación de los que ya conocemos. El siguiente programa es un ejemplo de cómo se calcula la media de una lista.

```
10 LET N=5 : DIM A(N)
20 FOR I=1 TO N
30    PRINT I
40    INPUT A(I)
50    NEXT I
60 LET S=0
70 FOR I=1 TO N
80    LET S=S+A(I)
90    NEXT I
100 LET M=S/N
110 PRINT "MEDIA=";M
```

Las líneas comprendidas entre la 10 y la 50 sirven para definir la list A y llenarla desde el teclado. En la línea 60 definimos la variable S y l inicializamos con un cero. A continuación se calcula la suma de los ele mentos mediante el bucle de las líneas 70, 80 y 90. Hasta aquí hemos re producido el procedimiento de suma de la lista. En la línea 100 calculamo la media y la almacenamos en M. Finalmente en la línea 110 se imprim el resultado.

Conviene remarcar que si se cambia el valor de N de la línea 10, e resto del programa funcionará de forma idéntica incluyendo el cálculo de l media.

Si en lugar de una lista tenemos una tabla, el cálculo de la media tien las tres posibilidades que vimos para el caso de la suma de los elementos Recordemos que estas posibilidades eran: media global, media de cad columna y media de cada fila. Utilizamos de nuevo la tabla de la figura como ejemplo.

El cálculo de las medias de una tabla requiere especificar la fila, la columna o el total

Para calcular la media aritmética global emplearemos el siguiente pro grama:

```
10 LET N=3 : LET M=2
20 DIM C(N,M)
30 FOR I=1 TO N
40    FOR J=1 TO M
50       INPUT C(I,J)
60       NEXT J
70    NEXT I
80 LET S=0
90 FOR I=1 TO N
100   FOR J=1 TO M
110      LET S=S+C(I,J)
120      NEXT J
130   NEXT I
140 LET X=S/(N*M)
150 PRINT "MEDIA=";X
```

La primera parte hasta la línea 130 ya la hemos estudiado y sirve par obtener la suma global de todos los elementos. En la línea 140 se evalú

la media que se almacena en X. Fijémonos que en una tabla, el número total de elementos es igual al producto de sus dimensiones. Por tanto, el valor de la suma S hay que dividirlo por dicho producto, en este caso N*M. Este producto está escrito entre paréntesis a causa del orden de prioridad de los operadores.

Entrando las notas en el orden 7, 8, 4, 6, 6 y 5 el resultado será 6.

Para calcular la media de una columna modificaremos el programa anterior a partir de la línea 80.

```
 80 LET S=0
 90 LET V=1
100 FOR I=1 TO N
110    LET S=S+C(I,V)
120    NEXT I
130 LET X=S/N
140 PRINT "MEDIA=";X
```

La variable V es la que establece el número de columna para el cual se calculará la media. Puesto que ahora sólo se suman los elementos de una columna, para calcular la media dividiremos únicamente por N (línea 130). Para la columna 1 la media es 5.666 y para la segunda es 6.333. Tal como indican las Matemáticas, la media de las dos medias anteriores es 6 y ha de coincidir con la media global.

Para calcular la media de cada fila utilizaremos el mismo procedimiento pero cambiando los subíndices. De nuevo, sólo modificaremos la última parte del programa. Esta última parte queda:

```
 80 LET S=0
 90 LET F=1
100 FOR J=1 TO M
110    LET S=S+C(F,J)
120    NEXT J
130 LET X=S/M
140 PRINT "MEDIA=";X
```

La variable F es la que establece el número de fila. En la línea 130, el valor de S se divide por M ya que cada fila tiene M elementos (en nuestro caso 2).

Las medias aritméticas de cada fila son 7.5, 5 y 5.5.

RESUMEN

El procedimiento básico para manipular los conjuntos dimensionados es modificar un elemento.

La instrucción LET es la que nos permite modificar un elemento del conjunto. Se coloca en la parte izquierda de la instrucción, es decir, antes de la asignación, el nombre de la variable del conjunto dimensionado y entre paréntesis el subíndice que queremos modificar.

En general, el método de seleccionar un elemento es válido para cualquier instrucción de BASIC. En INPUT, PRINT, cálculo, etc.

La manipulación de listas y tablas suele estar ligada inseparablemente a las instrucciones FOR/NEXT que se han estudiado en la lección anterior.

La razón de esta dependencia es que la instrucción FOR/NEXT está pensada para hacer acciones repetitivas y los conjuntos dimensionados son elementos todos iguales. Recuerde que la característica fundamental de los conjuntos dimensionados es la homogeneidad.

La utilización de la variable de control de un bucle para expresar el subíndice de un conjunto dimensionado conlleva una reducción de instrucciones y una mayor flexibilidad.

El esquema de manipulación de los conjuntos dimensionados está igualmente bien adaptado a la manipulación de variables textuales o numéricas.

El número de instrucciones FOR/NEXT asociadas a un conjunto dimensionado es normalmente el número de dimensiones.

Se puede recorrer una lista con un bucle, se recorre una tabla con dos bucles, etc.

La escritura de las listas y tablas se simplifica mucho con la utilización de los bucles y de la instrucción PRINT.

Otra técnica básica para la manipulación de listas y tablas es el rastreo de todos los elementos y mediante la instrucción de decisión (IF...THEN) realizar o no una acción con el elemento.

Un ejemplo consiste en encontrar el máximo de una tabla.

En la realización de un rastreo de la tabla se pueden realizar operaciones que nos den información resumida de la tabla, por ejemplo, cálculo de totales, cálculo de sumas de filas o sumas de columnas, o incluso hallar valores medios. En general utilizaremos variables que se van modificando convenientemente a medida que pasamos por cada elemento de la tabla.

La localización de los elementos de la tabla puede hacerse de dos maneras:

— Por posición. Es la manera más simple y está en la misma esencia de lo que son los conjuntos dimensionados. Se pide cuál es el contenido del elemento que está en una posición determinada.

— Por contenido. Se desea saber qué posición contiene un determinado valor. Es frecuente realizar este tipo de búsqueda con variables textuales pero también se da en variables numéricas.

La propiedad más importante es que el resultado puede ser que lo que se busca se encuentre o no se encuentre.

Este obliga a considerar la salida de un bucle FOR/NEXT mediante un proceso de decisión. Cuando se alcanza la salida normal del bucle es señal de que lo que se busca no se ha encontrado.

EJERCICIOS DE AUTOCOMPROBACION

Completar las frases siguientes:

26. La .... de un elemento es el procedimiento básico para manipular los conjuntos dimensionados.

27. Las instrucciones .... son el complemento inseparable para el tratamiento de los conjuntos dimensionados.

28. Los subíndices de los conjuntos dimensionados se relacionan con frecuencia con la variable de .... de un bucle.

29. El número de FOR/NEXT que necesitamos para recorrer una tabla es igual al número de .... del conjunto dimensionado.

30. La localización de los elementos de una tabla puede hacerse por posición o por ....

Encierre en un círculo la respuesta que corresponda a la alternativa correcta.

31. En el programa siguiente:

```
10 DIM A(5)
20 FOR I = 2 TO 5
30 LET A(I) = A(I−1)*2
40 NEXT I
```

qué instrucción precisa para que la tabla A contenga la secuencia 1, 2, 4, 8 y 16.

a) 15 LET A(3)=2

b) 15 LET A(1)=1

c) 35 LET A(3)=2

d) 35 LET A(1)=1

32. En el programa siguiente:

```
10 DIM A(5)
20 FOR I = 1 TO 5
30 LET A (I) = 2*I-4
40 NEXT I
50 LET S = 0
60 FOR I = 1 TO 5
70 LET S = S + A(I)
80 NEXT I
90 PRINT S/5
```

¿Qué número escribe?

a) 1
b) 2
c) 3
d) 4

33. Sea el programa siguiente:

```
10 DIM A(3,3)
20 FOR I = 1 TO 3
30 FOR J = 1 TO 3
40 LET A(I,J) = I + J
50 NEXT I
60 NEXT J
```

¿Cuál es el valor de la tabla resultante? (Añádale las instrucciones que faltan para que le dé el resultado en pantalla en forma de tabla).

a)

2	3	4
5	6	7
8	9	10

b)

2	3	4
3	4	5
4	5	6

c)

2	4	3
3	5	4
4	6	5

d)

1	2	5
5	2	1
2	5	1

(Se supone que la primera dimensión son las filas y la segunda las columnas).

34. Si en el programa del ejercicio anterior se sustituye la línea 40 por

```
40 LET A(I,J) = I−J
```

¿Cuál es el valor de la tabla?

a)

0	2	1
−2	0	3
−1	−3	0

b)

2	3	4
5	6	7
8	9	10

c)

−1	−2	0
−2	0	−1
0	−1	0

d)

0	−1	−2
1	0	−1
2	1	0

(Se supone que la primera dimensión son las filas y la segunda las columnas).

35. En el programa siguiente:

```
10 DIM A(3)
20 LET A(1)=2:LET A(2)=1:LET A(3)=3
30 FOR I= 1 TO 3
40 FOR J= I+1 TO 3
50 IF A(I) <= A(J) THEN GO TO 70
60 LET T=A(I) : LET A(I)=A(J) : LET A(J)=T
70 NEXT J
80 NEXT I
```

Después de ejecutar el RUN ¿Cuál es el valor de la lista A?

a) 2,1,3
b) 3,2,1
c) 3,1,2
d) 1,2,3



Capítulo 11
• Almacenamiento de los datos en el programa

ESQUEMA DE CONTENIDO

Objetivos	
Cómo almacenar datos en un programa	La instrucción DATA La instrucción READ La instrucción RESTORE
Aplicaciones	El mes en letras El día en letras La agenda
Funciones definidas por el usuario	Nomenclatura Definición de las funciones Utilización de las funciones

11.0 OBJETIVOS

Este capítulo sigue con las instrucciones del BASIC que no son indispensables para programar pero que dan una comodidad, que a veces se convierte en necesidad.

En la primera parte, se trata el problema de almacenar datos que son fijos.

Muchas veces, en las aplicaciones deseamos tener un conjunto de datos memorizados que permiten eliminar la engorrosa tarea de la entrada cada vez que ejecutamos el programa.

Esto sucede en tablas de nombres de artículos, precios, listas de alumnos y así un sinfín de posibilidades.

El BASIC dispone de un mecanismo para almacenar esta información en el propio programa sin tener que utilizar la instrucción LET cada vez.

Estas instrucciones son el DATA, el READ y el RESTORE.

El punto más importante es comprender el acceso a estos datos memorizados, pues tiene unas características cuyo conocimiento es imprescindible para hacerlo funcionar correctamente. Además, este mecanismo tiene aplicaciones más amplias, que estudiaremos en capítulos posteriores.

Los ejemplos tratan el problema de la búsqueda por nombre. Tenga en cuenta que el 90 % de la programación consiste en buscar cosas en tablas.

En conclusión, aunque el capítulo está dedicado a enseñar la utilización de las instrucciones DATA, READ y RESTORE contiene el embrión de métodos muy generales en informática.

Ponga atención en la forma de utilizar las instrucciones y también en los métodos utilizados en los ejemplos.

En la segunda parte se introduce una herramienta que resulta muy útil en programas en que un cálculo relativamente complejo se utiliza muchas veces. Se trata de las funciones definidas por el programador.

Las funciones intrínsecas que hemos visto en el capítulo 3 del primer tomo permiten hacer cálculos complicados con comodidad.

¿Ha considerado alguna vez cómo tendría que hacer un programa que sacara la raíz cuadrada sin utilizar la función SQR?

Si reflexiona un poco verá que no es sencillo.

Cuando se nos presentan estos cálculos complicados y repetitivos es cómodo poder definir funciones que nos den el resultado sin tener que repetir la escritura de las operaciones que lo originan en muchos sitios del programa.

Estudie con detalle este mecanismo, pues su repercusión es mucho más amplia que la que pone de manifiesto el BASIC.

Hay lenguajes de programación que basan toda su potencia precisamente en la definición de funciones por el propio programador.

La comprensión de las funciones definidas, aparte de utilizar este recurso del BASIC, le servirán para entender ciertos puntos de la ciencia informática.



11.1 COMO ALMACENAR DATOS EN UN PROGRAMA

Hemos visto dos maneras de asignar valores a una variable dentro de un programa. Para ello, conocemos dos instrucciones, que son la instrucción INPUT y la instrucción LET.

Como ya sabemos, la instrucción INPUT nos permite introducir —al tiempo de ejecución— el valor que debe tomar la variable. Este valor puede ser distinto cada vez que se ejecute el programa.

Por otra parte, la instrucción LET sirve para asignar un valor fijo a una variable.

La elección entre ambas instrucciones dependerá de las posibilidades de variación del valor a asignar.

Vamos a verlo mediante un ejemplo. Supongamos que necesitamos un programa para calcular la media de las notas de los alumnos de una clase. Supondremos que la clase consta únicamente de 5 alumnos, para evitar que el programa se alargue demasiado. Sin embargo, veremos que el programa resulta fácilmente adaptable a un número cualquiera de alumnos (mayor o menor).

Datos fijos y datos variables

En el programa, hay una serie de datos fijos (el nombre de los alumnos, que a lo largo de todo un curso es el mismo) y un conjunto de datos variables (las notas obtenidas en cada examen).

El programa estará constituido por varias partes. En primer lugar, deberá leer y asignar a una variable el nombre de cada alumno, y permitir que el usuario introduzca la nota correspondiente a cada uno de ellos.

Por otra parte, una vez conocidas todas las notas, realizaremos el cálculo de la media, y por último, escribiremos la lista completa con las notas y el resultado obtenido.

De cara a manipular los nombres con comodidad, los asignaremos a un conjunto dimensionado, que en este caso constará de 5 elementos. Lo mismo haremos con los valores de las notas. El programa para ello será:

```
NEW
10 REM LISTA DE NOTAS Y DETERMINACION DE LA MEDIA
20 REM Asignacion del numero de alumnos (NA)
30    LET NA = 5
40 REM Dimensionado del conjunto alumnos (N$) y notas (T)
50    DIM N$(NA), T(NA)
60 REM Lectura de los nombres
70    LET N$(1)="Fernandez"
80    LET N$(2)="Garcia"
90    LET N$(3)="Lopez"
100   LET N$(4)="Rodriguez"
110   LET N$(5)="Sanchez"
120 REM Entrada de las notas de cada alumno
130   LET S=0 : CLS
140   FOR I = 1 TO NA
150      PRINT "Entre la nota de ";N$(I);
160      INPUT T(I)
170      LET S = S + T(I)
180      NEXT I
190 REM Escritura de resultados
200   CLS : PRINT "LISTA DE NOTAS" : PRINT
210   FOR I = 1 TO NA
220      PRINT N$(I);" ";T(I)
```

```
230          NEXT I
240      PRINT : PRINT "LA MEDIA ES: ";S/NA
250 END
```

El funcionamiento del programa es el siguiente: una vez conocidos los nombres de cada alumno —líneas 70 a 110—, se introduce la nota correspondiente —líneas 140 a 180— y simultáneamente se realiza la suma de cada una de las notas que se van introduciendo, en un acumulador (S) que previamente se habrá igualado a cero —línea 130—. De esta forma, preparamos el cálculo posterior de la media, que se obtendrá sumando las notas de cada alumno y dividiendo por el número total de alumnos. La última fase, tal como se indica en el propio programa, corresponde a la impresión de la lista completa (nombre de cada alumno, junto con la nota obtenida) así como del valor de la media.

De esta forma, cada vez que se deba confeccionar una lista bastará con introducir las notas, ya que dentro del propio programa mantenemos los nombres almacenados, y en orden alfabético.

Dificultad en mantener el orden cuando hay una adición

Sin embargo, si el número de alumnos fuera muy grande (50, 60, 100, ...), necesitaríamos un gran número de instrucciones de asignación, y además, cada vez que se tuviera que intercalar un elemento, para mantener el orden alfabético, se deberían modificar todas estas instrucciones, a partir de la que se haya debido intercalar. Por ejemplo, supongamos que hay que añadir el nombre de un alumno que empiece por H, por ejemplo, Hernández. Este nombre deberá ocupar la tercera posición de la lista. Habrá que intercalar, pues, la instrucción:

```
85      LET N$(3)="Hernandez"
```

y modificar las restantes a partir de ésta, que deberán quedar:

```
 90      LET N$(4)="Lopez"
100      LET N$(5)="Rodriguez"
110      LET N$(6)="Sanchez"
```

Hay que modificar también la línea 30, ya que el número de alumnos es ahora de seis. Por tanto quedará ahora:

```
30   LET NA = 6
```

El listado completo será pues:

```
10 REM LISTA DE NOTAS Y DETERMINACION DE LA MEDIA
20 REM Asignacion del numero de alumnos (NA)
30    LET NA = 6
40 REM Dimensionado del conjunto alumnos (N$) y notas (T)
50    DIM N$(NA), T(NA)
60 REM Lectura de los nombres
```

```
 70     LET N$(1)="Fernandez"
 80     LET N$(2)="Garcia"
 85     LET N$(3)="Hernandez"
 90     LET N$(4)="Lopez"
100     LET N$(5)="Rodriguez"
110     LET N$(6)="Sanchez"
120 REM Entrada de las notas de cada alumno
130     LET S=0 : CLS
140     FOR I = 1 TO NA
150         PRINT "Entre la nota de ";N$(I);
160         INPUT T(I)
170         LET S = S + T(I)
180         NEXT I
190 REM Escritura de resultados
200     CLS : PRINT "LISTA DE NOTAS" : PRINT
210     FOR I = 1 TO NA
220         PRINT N$(I);"  ";T(I)
230         NEXT I
240     PRINT : PRINT "LA MEDIA ES: ";S/NA
250 END
```

Esto ya resulta engorroso, pero si el número de alumnos es grande cualquier modificación de la lista puede resultar muy pesada.

Por esta razón, existe en BASIC un mejor sistema de mantener datos almacenados en el programa. Para ello se utilizan dos instrucciones que vamos a conocer a continuación. Estas instrucciones son la instrucción *DATA* y la instrucción *READ*.

En los ejemplos que veremos para conocer la aplicación de estas instrucciones, utilizaremos conceptos que hemos visto en secciones anteriores.



11.1.1 La instrucción DATA

Esta instrucción sirve para almacenar datos dentro de un programa. Se escribe la palabra DATA, y a continuación la lista de datos a almacenar, separados por comas.

Esta palabra es inglesa, aunque de origen latino y corresponde al plural latino de la palabra «datos».

Estos datos pueden ser numéricos o alfanuméricos, es decir, podemos almacenar números o palabras, indistintamente y en cualquier orden.

Las instrucciones DATA pueden escribirse en cualquier parte del programa, al principio, al final o intercaladas con las restantes instrucciones.

Sin embargo, de cara a una mejor organización, y para hacer que el programa sea más legible, suelen situarse todas juntas al final del mismo. Se pueden escribir tantas instrucciones DATA como sean necesarias.

Cada instrucción DATA —como todas las instrucciones de BASIC— puede tener una longitud máxima de 255 caracteres. Sin embargo, no es necesario escribir todos los datos en una sola instrucción. Podemos escribirlos en varias instrucciones, tantas como sean necesarias, ya que no hace falta que cada instrucción DATA tenga la máxima longitud.

El orden de los datos debe mantenerse

Sin embargo, sí que debe mantenerse —se escriban los datos donde se escriban—, el orden de los mismos. El almacén se puede considerar continuo desde la primera hasta la última instrucción DATA. Para leer un valor situado en un punto determinado de la lista, habrá que leer todos los valores situados antes que él en el almacén.

Veamos algunos ejemplos. Supongamos que queremos almacenar tres datos cualquiera, por ejemplo los tres números siguientes: 25, 3 y 2567. Para ello escribiremos:

```
10 DATA 25, 3, 2567
```

Dos tipos de los datos pueden mezclarse

Si quisiéramos almacenar tres palabras, por ejemplo DINOSAURIO, AUTOBUS y PARAGÜERO, podríamos escribir:

```
10 DATA "DINOSAURIO", "AUTOBUS", "PARAGUERO"
```

No hace falta que los datos que se almacenan mediante instrucciones DATA sean del mismo tipo, sino que pueden tener datos numéricos y alfanuméricos mezclados. Así, podemos almacenar los números y las palabras de los dos ejemplos anteriores mezclados, por ejemplo de la siguiente forma:

```
10 DATA 25,"DINOSAURIO",3,"AUTOBUS",2657,"PARAGUERO"
```

Observe que hemos escrito los datos alfanuméricos entre comillas, y los datos numéricos sin ellas. Hemos escrito los datos en un orden determinado, que hemos escogido al azar. Insistimos en que el orden en que se almacenan los datos puede ser cualquiera, pero debe tenerse muy en cuenta en el momento de leerlos, de forma que por una parte coincida el tipo de variable que se lee con el tipo de dato almacenado, y por otra parte, sepamos en cada momento la correspondencia entre lo que está almacenado, y el lugar que ocupará en el programa cuando se lea, es decir, la variable a la que se va a asignar.

Si quisiéramos almacenar los cinco nombres de los alumnos de la clase en el programa para confeccionar listas de notas que hemos visto antes podremos hacerlo, ya sea de esta forma:

```
1000 DATA "Fernandez", "Garcia", "Rodriguez", "Lopez", "Sanchez"
```

o bien:

```
1000 DATA "Fernandez"
1010 DATA "Garcia"
```

```
1020 DATA "Lopez"
1030 DATA "Rodriguez"
1040 DATA "Sanchez"
```

O bien con cualquier otra forma de agrupar los nombres (dos, dos y uno, o bien tres y dos...). Lo importante es mantener siempre el orden correcto, que en este caso debe ser el orden alfabético.

Veamos ahora cómo leer los datos almacenados.

11.1.2 La instrucción READ

Esta instrucción sirve para leer los datos almacenados mediante una o varias instrucciones DATA. Se escribe la palabra READ, y a continuación la lista de variables a las que se van a asignar los valores almacenados, separados por comas.

READ significa «leer»

A la lista de variables que sigue a la palabra READ, se le asigna uno a uno, los valores contenidos en el almacén. Pueden tenerse tantas instrucciones READ como se quiera, es decir, no es necesario leer de una sola vez todos los valores almacenados, sino que cada vez que se encuentra una instrucción READ seguida de una lista de variables, se leen tantos valores como número de variables contenidas en la instrucción READ.

Una vez que se ha asignado un valor, el ordenador lo «retira» del almacén, y a la próxima variable situada en una instrucción READ, se le asignará el valor siguiente al último leído.

Deben tenerse en cuenta dos cosas muy importantes:

Fuentes de error en la utilización del READ

1a) El tipo de variables ha de coincidir con el tipo de valor almacenado.

- Si el valor es alfanumérico la variable debe ser textual.
- Si el valor es numérico, la variable podrá ser numérica o textual. Evidentemente si la variable es textual, no se toma el valor numérico como tal, sino como texto.

2a) El número de elementos almacenados ha de coincidir —o en todo caso ser superior— al número de variables que siguen a una instrucción READ. Los valores se asignarán uno a uno a cada variable a leer. Si hay más valores en el almacén, no se produce ningún problema, simplemente no se asignan los valores que sobran. Sin embargo, si el número de elementos almacenados es menor, se produce un error y se detiene el programa.

En cierta forma, podemos considerar la instrucción READ semejante a la instrucción INPUT. La diferencia entre ambas reside en el lugar de donde se lee el valor de la variable. Cuando hacemos un INPUT, el programa asigna a la variable el valor que le introduzcamos por el teclado. Cuando

hacemos un READ, el valor asignado a la variable es el que corresponda del almacén.

Veamos cómo leer e imprimir valores que tengamos almacenados.

En primer lugar veremos cómo leer los tres números: 25, 3 y 2567 que habíamos almacenado mediante una instrucción DATA, y una vez leídos, los imprimiremos. El programa para ello será:

```
NEW
10 DATA 25, 3, 2567
20 READ N1, N2, N3
30 PRINT N1, N2, N3
```

Podremos ver que aparecen los tres números escritos en la pantalla.

Supongamos ahora que añadimos una nueva instrucción DATA, con las tres palabras almacenadas: DINOSAURIO, AUTOBUS y PARAGÜERO, sin modificar el programa de momento. Si escribimos por ejemplo:

```
15 DATA "DINOSAURIO", "AUTOBUS", "PARAGUERO"
```

y ejecutamos el programa, veremos que de nuevo aparecen en pantalla los mismos números que en el caso anterior. Esto es así, porque sólo leemos tres valores (los que se leen en la instrucción 20), y los otros tres —correspondientes a la segunda instrucción DATA, que contiene los nombres— no se leen. Si deseamos leerlos también y escribirlos, añadiremos dos instrucciones. La primera será:

```
25 READ A$, B$, C$
```

y la segunda será:

```
35 PRINT A$, B$, C$
```

El programa quedará ahora:

```
10 DATA 25, 3, 2567
15 DATA "DINOSAURIO", "AUTOBUS", "PARAGUERO"
20 READ N1, N2, N3
25 READ A$, B$, C$
30 PRINT N1, N2, N3
35 PRINT A$, B$, C$
```

Ejecutando ahora el programa, veremos en pantalla:

25	3	2567
DINOSAURIO	AUTOBUS	PARAGÜERO

De esta forma se leerán los seis valores almacenados y se escribirán, primero los tres números y después las tres palabras.

Si los hubieramos almacenado mezclados, deberíamos leerlos también mezclados. Así, por ejemplo, podríamos tener:

```
NEW
10 DATA 25, "DINOSAURIO", 3
20 DATA "AUTOBUS", 2567, "PARAGUERO"
30 READ N1, A$
40 READ N2, B$
50 READ N3, C$
60 PRINT  N1; A$; N2; B$; N3; C$
```

Observe el efecto de este programa ejecutándolo. Aparecerá escrito en pantalla:

```
25 DINOSAURIO 3 AUTOBUS 2567 PARAGUERO
```

Es importante ver que hemos utilizado más de una instrucción DATA para almacenar los datos, y que vamos leyéndolos conforme los necesitamos, independientemente de como estén almacenados. Así, la primera instrucción READ lee los dos primeros valores del almacén —un número y una palabra—, la segunda los dos siguientes, y la tercera los dos últimos. Aunque hemos escrito tres valores —número, palabra y número— en la primera instrucción DATA y tres más en la segunda —palabra, número, palabra—.

Concepto de lectura secuencial

Es en este sentido en el que decimos que el almacén es continuo, aunque lo hayamos escrito utilizando varias instrucciones DATA, y que se lee de forma secuencial: un dato a continuación del otro.

Vamos a ver con el ejemplo de la lista de alumnos de una clase, la forma de utilizar las instrucciones DATA y READ. El listado del programa será en este caso:

```
NEW
10 REM LISTA DE NOTAS Y DETERMINACION DE LA MEDIA
20 REM Asignacion del numero de alumnos (NA)
30    LET NA = 5
40 REM Dimensionado del conjunto alumnos (N$) y notas (T)
50    DIM N$(NA), T(NA)
60 REM Lectura de los nombres
70    FOR I = 1 TO NA
```

```
  80         READ N$(I)
  90         NEXT I
 120 REM Entrada de las notas de cada alumno
 130     LET S=0 : CLS
 140     FOR I = 1 TO NA
 150         PRINT "Entre la nota de ";N$(I);
 160         INPUT T(I)
 170         LET S = S + T(I)
 180         NEXT I
 190 REM Escritura de resultados
 200     CLS : PRINT "LISTA DE NOTAS" : PRINT
 210     FOR I = 1 TO NA
 220         PRINT N$(I);" ";T(I)
 230         NEXT I
 240     PRINT : PRINT "LA MEDIA ES: ";S/NA
 250 END
1000 DATA "Fernandez"
1010 DATA "Garcia"
1020 DATA "Lopez"
1030 DATA "Rodriguez"
1040 DATA "Sanchez"
```

En este caso, podemos hacer la lectura mediante un bucle, con lo cual reducimos a tres el número de instrucciones, ya que evitamos todas las asignaciones.

El funcionamiento de un bucle ya lo conocemos. En primer lugar se asignará a la variable I el valor inicial. A continuación se ejecutará la instrucción READ. Dado que es la primera vez que se ejecuta, asignará como primer elemento de la variable N$, el primer valor almacenado en la primera instrucción DATA del programa. En este caso este valor es el apellido Fernández, tendremos pues:

```
LET N$(1) = "Fernandez"
```

La instrucción NEXT nos remite al principio del bucle de nuevo. Deberá leerse ahora el valor de N$(2). El programa tomará ahora el segundo dato del almacén —García—; por tanto:

```
LET N$(2) = "Garcia"
```

Bucle para la lectura del almacén en DATA

El bucle se irá ejecutando de esta forma hasta asignar los cinco valores a la variable N$.

Vemos pues, que se han reducido el número de instrucciones de asignación. De esta forma, con un bucle de este estilo, podremos realizar la asignación de todos los elementos, sea cual sea el número de éstos.

Por otra parte, hemos añadido las instrucciones que contienen los datos. Estas instrucciones suelen numerarse con valores más altos que los que les correspondería en orden correlativo, de cara a posibles modificaciones que se puedan producir en el programa. De esta forma, aunque el programa se haga más largo, el almacén queda siempre situado al final del mismo, y agrupado.

Por otra parte, hemos escrito los nombres de uno en uno, pues si hay que intercalar un nombre —por ejemplo Hernández— bastará añadir una instrucción DATA con el número de línea correspondiente. En este caso, escribiríamos:

```
1015 DATA "Hernandez"
```

Modificando ahora el número total de alumnos, NA, ya podremos utilizar el programa para efectuar listas. Así, deberemos variar la línea 30:

```
30    LET NA=6
```

El listado completo quedará ahora:

```
  10 REM LISTA DE NOTAS Y DETERMINACION DE LA MEDIA
  20 REM Asignacion del numero de alumnos (NA)
  30    LET NA = 6
  40 REM Dimensionado del conjunto alumnos (N$) y notas (T)
  50    DIM N$(NA), T(NA)
  60 REM Lectura de los nombres
  70    FOR I = 1 TO NA
  80       READ N$(I)
  90       NEXT I
 120 REM Entrada de las notas de cada alumno
 130    LET S=0 : CLS
 140    FOR I = 1 TO NA
 150       PRINT "Entre la nota de ";N$(I);
 160       INPUT T(I)
 170       LET S = S + T(I)
 180       NEXT I
 190 REM Escritura de resultados
 200    CLS : PRINT "LISTA DE NOTAS" : PRINT
 210    FOR I = 1 TO NA
 220       PRINT N$(I);" ";T(I)
 230       NEXT I
 240    PRINT : PRINT "LA MEDIA ES: ";S/NA
 250 END
1000 DATA "Fernandez"
1010 DATA "Garcia"
1015 DATA "Hernandez"
1020 DATA "Lopez"
1030 DATA "Rodriguez"
1040 DATA "Sanchez"
```

Compruebe el funcionamiento del programa ejecutándolo.

Hemos visto hasta ahora que cuando se ha de almacenar un valor alfanumérico lo hemos escrito siempre entre comillas. Algunos modelos de ordenadores permiten suprimirlas, es decir, que se pueden escribir los nombres omitiendo las comillas. Si esto es posible en su ordenador particular, ya lo sabe por el capítulo de prácticas.

La supresión de las comillas

Si la máquina lo permite, escribir los nombres sin comillas puede resultar práctico; sin embargo debe tenerse en cuenta que esto no se puede hacer en todos los casos. Veamos un ejemplo.

Supongamos ahora, que deseamos el nombre completo de cada alumno, es decir, el primer apellido, el segundo apellido y el nombre.

Los nombres suelen escribirse en estos casos de esta forma:

1.er Apellido 2.° Apellido, Nombre

En principio, bastará añadir los correspondientes datos en la lista del almacén. Deberemos, pues, modificar las instrucciones desde la línea 1000 a la 1040. Quedaría por ejemplo:

```
1000 DATA "Fernadez Prats, Javier"
1010 DATA "Garcia Valls, Elena"
1015 DATA "Hernadez Ros, Pedro"
1020 DATA "Lopez Juan, Ana"
1030 DATA "Rodriguez Cots, Olga"
1040 DATA "Sanchez Robles, Miguel"
```

Pero suponiendo que la máquina lo permitiera, si en este caso no escribiéramos los nombres entre comillas, se produciría un problema a la hora de interpretar cada valor por la máquina, ya que, como sabemos, las comas sirven para separar cada uno de los valores almacenados. Por esta razón, el primer nombre que la máquina tomaría sería Fernández Prats, el siguiente sería Javier, el siguiente García Valls, etc.

Los blancos al principio de un dato

Por tanto, cuando se desean incluir comas en el texto, se debe escribir éste entre comillas, aun en el caso de que la máquina permita omitirlas. Lo mismo sucede si se quieren escribir blancos al principio del texto.

En todo caso, debe tenerse en cuenta que el programa intentará leer y asignar los datos del almacén siguiendo el proceso lógico, mientras le sea posible.

Por otra parte, en este ejemplo, hemos utilizado una variable dimensionada para almacenar los datos, ya que nos resulta más práctico. Sin embargo, puede hacerse la lectura de los valores almacenados en un DATA utilizando variables con nombres distintos. Así vamos a verlo con un ejemplo. Construiremos un programa traductor muy simple, de forma que al escribir una palabra en castellano, nos dé la correspondiente en inglés, o un mensaje indicativo, si no conoce la palabra.

Un listado para este caso podría ser:

```
NEW
10 REM TRADUCTOR CASTELLANO-INGLES
20 REM Introduccion de la palabra
30   INPUT "Que palabra desea traducir? ",P$
40 REM Traduccion
50   READ E$, T$
60   IF E$="FIN" THEN GOTO 100
70   IF P$<>E$ THEN GOTO 50
80   PRINT "La palabra ";P$;" se traduce por ";T$
```

```
  90   GOTO 110
 100   PRINT "No conozco esta palabra"
 110 END
1000 DATA "HOLA", "HELLO"
1010 DATA "ADIOS", "GOOD BYE"
1020 DATA "BUENOS DIAS", "GOOD MORNING"
1030 DATA "LAPIZ", "PENCIL"
1040 DATA "MESA", "TABLE"
1050 DATA "FIN",""
```

Observe el funcionamiento de este programa. En el almacén existe un número de palabras traducidas, en este caso únicamente cinco, sin embargo, sin efectuar ninguna modificación adicional en el programa, podríamos alargar el almacén tanto como quisiéramos, mientras nos lo permitiera la memoria de la máquina.

Esto se puede hacer por la forma en que hemos construido el programa. El funcionamiento de éste es muy simple. Una vez introducida la palabra a traducir, el programa empieza a buscarla en el almacén. La búsqueda se realiza de forma lineal, leyendo de dos en dos los valores almacenados. El primero que leemos corresponde a la palabra en castellano (E$), y el segundo es la traducción de la misma en inglés (T$). Esta búsqueda finaliza por dos razones, o bien porque se ha encontrado la palabra que se busca, con lo que se escribe ésta y su traducción, o bien porque se han leído todas las palabras del almacén.

Para indicar el final del mismo, se escribe como valor último la palabra FIN, y cada vez que se efectúa una lectura, se comprueba si se ha llegado al final. Si es así, es que no hemos encontrado la palabra, con lo que damos el mensaje correspondiente.

El marcador del final del almacén

Nótese que después de la palabra FIN hemos escrito un dato nulo. Lo hacemos así porque cada vez que efectuamos una lectura, necesitamos dos valores. Si no lo escribiéramos, y termináramos con la palabra FIN, tendríamos un error en la última lectura pues nos faltaría un valor, ya que leemos los valores de dos en dos, y necesitamos siempre un número par de éstos.

De esta manera, podríamos conocer la traducción de cualquier palabra, excepto la de la propia palabra FIN. Si quisiéramos que el programa la incluyera, deberíamos utilizar otra marca de final. Por ejemplo, usando el asterisco (*). Así, podríamos modificar la línea 60:

```
60    IF E$="*" THEN GOTO 100
```

y la 1050:

```
1050 DATA "FIN", "END"
```

Finalmente deberíamos añadir la nueva marca de final:

```
1060 DATA "*", ""
```

El programa quedará ahora:

```
NEW
  10 REM TRADUCTOR CASTELLANO-INGLES
  20 REM Introduccion de la palabra
  30   INPUT "Que palabra desea traducir? ", P$
  40 REM Traduccion
  50   READ E$, T$
  60   IF E$="*" THEN GOTO 100
  70   IF P$<>E$ THEN GOTO 50
  80   PRINT "La palabra ";P$;" se traduce por ";T$
  90   GOTO 110
 100   PRINT "No conozco esta palabra"
 110 END
1000 DATA "HOLA", "HELLO"
1010 DATA "ADIOS", "GOOD BYE"
1020 DATA "BUENOS DIAS", "GOOD MORNING"
1030 DATA "LAPIZ", "PENCIL"
1040 DATA "MESA", "TABLE"
1050 DATA "FIN", "END"
1060 DATA "*" ""
```

En este caso hemos escogido un carácter —que puede ser cualquiera— de forma que no interfiera con las palabras a traducir. Ejecute ahora el programa, y si introduce la palabra FIN, obtendrá la correspondiente traducción.

Parecería útil, por otra parte, construir el programa de forma que cada vez que se ejecute se puedan traducir tantas palabras como se desee, sin necesidad de realizar una nueva ejecución para cada palabra.

Podríamos modificar el programa de forma que sea el propio usuario el que dé la indicación de continuar o finalizar. Para ello bastaría añadir o modificar las instrucciones:

```
110     INPUT "DESEA CONTINUAR (S/N): ", R$
120     IF R$="S" THEN GOTO 30
130 END
```

De esta forma, una vez realizada la búsqueda de una palabra, el propio programa permitiría continuar.

Realice la prueba introduciendo en primer lugar una palabra que sepa seguro que no está en el almacén, por ejemplo DINOSAURIO. Observará que el programa indica que no conoce la traducción de esta palabra, y a continuación le pregunta si desea continuar.

Pulse una «S» para responder afirmativamente, y escriba otra palabra

cualquiera. Inmediatamente la máquina le dará error, y se detendrá el proceso.

¿A qué se debe esto? Intente pensar la respuesta antes de seguir leyendo.

Se ha alcanzado el final del DATA y posteriormente se comete el error de leer

Si recuerda lo que hemos dicho del funcionamiento de las instrucciones READ, encontrará la razón de lo sucedido. Veámoslo paso a paso. Cuando ejecutamos el programa por primera vez, el programa empieza a buscar la palabra introducida, para lo cual va recorriendo el almacén, leyendo los valores de dos en dos. Dado que la palabra que hemos escrito (DINOSAURIO) no se encuentra en el «diccionario» que tenemos construido, el programa recorre todo el almacén hasta encontrar y leer el símbolo «*» que marca el final del mismo. En este punto nos da el mensaje correspondiente, y nos pide si deseamos continuar. Al responder nosotros afirmativamente, nos pregunta la nueva palabra a buscar, e intenta hacer una nueva lectura en el almacén. Pero, tal como hemos visto, en la vuelta anterior ya lo había recorrido todo, por tanto, no encuentra el siguiente valor a leer, así que nos da el mensaje de error correspondiente y se detiene.

¿Cómo podemos solucionar este problema? Evidentemente podríamos escribir RUN —ejecutar el programa— cada vez que deseemos conocer una palabra, pero esto no parece muy práctico. En la siguiente sección veremos una instrucción que nos permitirá solucionar este problema.

11.1.3 La instrucción RESTORE

Esta instrucción sirve para reinicializar la lectura del almacén, es decir, situarnos en las condiciones iniciales, de forma que al ejecutar una instrucción READ, sea cual sea el último valor leído, asigne a la variable correspondiente el primer valor almacenado en el mismo.

La palabra RESTORE se puede traducir por «restaurar» las condiciones iniciales, «reiniciar».

Semejanza de las instrucciones DATA, READ y RESTORE a una cinta magnética

En un sentido estricto, significa rebobinar, refiriéndonos a una cinta magnetofónica. Si consideramos la estructura y el funcionamiento de las instrucciones DATA, veremos que es semejante al de aquélla.

En primer lugar, hemos dicho que el almacén constituido por las instrucciones DATA se pueden considerar continuo, desde la primera hasta la última instrucción DATA. En segundo lugar, vemos que no podemos acceder a un punto intermedio de la misma sin pasar previamente toda la cinta situada antes que él. Además, no podemos seguir leyendo una vez se ha llegado al final. Para volver a leer debemos rebobinar. Este es el sentido de la instrucción RESTORE. En este aspecto, hace referencia a la vuelta a las condiciones iniciales, la vuelta al principio del almacén.

La situación de esta instrucción dentro del programa es un hecho importante para manipular correctamente los datos almacenados. De hecho, todas las instrucciones deben estar en su sitio para que el programa trabaje armónicamente.

Veamos con un ejemplo cuál es el efecto de situar incorrectamente esta instrucción. Escribámosla para ello entre las líneas 50 y 60:

```
55    RESTORE
```

Ejecutemos ahora el programa, y escribamos una palabra cualquiera, por ejemplo DINOSAURIO. Observaremos que el programa cae en un bucle sin final; para recuperar el control deberemos pulsar la tecla de interrupción correspondiente.

¿A qué se debe lo sucedido?

Una vez hemos introducido la palabra el problema efectúa la lectura del almacén. Dado que es la primera vez que ejecutamos la instrucción READ, se leerá el primer par de valores (en este caso HOLA y HELLO). A continuación se ejecuta la instrucción RESTORE, cuyo efecto no se nota momentáneamente. Dado que la palabra leída en el almacén es distinta de la marca de final, y que además es distinta de la palabra introducida, el programa nos remite de nuevo a la línea 50. En este punto se efectúa una nueva lectura del almacén. ¿Pero qué valores son los que se leen? Dado que hemos ejecutado previamente una instrucción RESTORE, los valores que toca leer son de nuevo los primeros del almacén (HOLA y HELLO). Ya que no hemos encontrado el fin del almacén y que sigue sin ser igual a la palabra introducida, volveremos de nuevo a la línea 50, habiendo ejecutado naturalmente la instrucción RESTORE situada en la línea 55 y así seguirá en un proceso sin fin.

De esta manera, no tenemos posibilidad de salir del círculo en el que hemos entrado, y nos vemos obligados a forzar la interrupción del mismo.

Sin embargo, situando la instrucción en el lugar adecuado, podremos utilizar el programa sin problemas para traducir todas las palabras que queramos. Así, borraremos la línea 55, y escribiremos:

```
115   RESTORE
```

El listado completo nos quedará ahora:

```
  10 REM TRADUCTOR CASTELLANO-INGLES
  20 REM Introduccion de la palabra
  30   INPUT "Que palabra desea traducir? ",P$
  40 REM Traduccion
  50   READ E$, T$
  60   IF E$="*" THEN GOTO 100
  70   IF P$<>E$ THEN GOTO 50
  80   PRINT "La palabra ";P$;" se traduce por ";T$
  90   GOTO 110
 100   PRINT "No conozco esta palabra"
 110   INPUT "Desea continuar? (S/N): ", R$
 115   RESTORE
 120   IF R$="S" THEN GOTO 30
 130 END
1000 DATA "HOLA", "HELLO"
1010 DATA "ADIOS", "GOOD BYE"
1020 DATA "BUENOS DIAS", "GOOD MORNING"
1030 DATA "LAPIZ", "PENCIL"
1040 DATA "MESA", "TABLE"
1050 DATA "FIN", "END"
1060 DATA "*",""
```

Compruebe ahora el correcto funcionamiento del programa.

RESUMEN

Los datos que utiliza un programa son fijos o variables. Se entienden por datos fijos aquellos que no varían con el tiempo, por ejemplo, los nombres de los alumnos.

Se entienden por datos variables aquellos que varían cada vez que ejecutamos el programa, por ejemplo, las notas.

Evidentemente no hay ningún dato absolutamente fijo, los nombres de los alumnos varían de curso en curso.

Sin embargo, se puede decir que son fijos porque tienen el mismo valor al ejecutarse muchas veces el mismo programa.

Una técnica para introducir estos datos fijos es mediante el INPUT; tiene los inconvenientes de tener que teclear cada vez listas más o menos largas y sobre todo es posible equivocarse.

Otra manera es colocar instrucciones LET para asignar estos valores fijos. La técnica es engorrosa cuando hay que añadir algún nuevo elemento y, en general, obliga a modificar muchos más elementos debido a que deseamos que guarden un orden determinado.

El BASIC tiene una serie de instrucciones preparadas para manipular estos casos. Son tres: el DATA, el READ y el RESTORE.

La instrucción DATA sirve para almacenar los datos en el propio programa.

Su formato es el nombre DATA seguido de uno o más datos separados por comas. El límite viene impuesto por la longitud de la línea únicamente.

Los datos dentro de una instrucción DATA pueden ser de cualquier tipo y estar mezclados.

Las instrucciones DATA pueden colocarse en cualquier parte del programa aunque es recomendable hacerlo al final o al principio. Si los datos no caben en una sola instrucción pueden utilizarse varias. En este caso, es importante que el orden de la numeración de las líneas siga el orden que establecemos para los datos.

Este orden es importante pues para acceder a uno determinado es necesario leer todos los anteriores.

La instrucción READ es la que permite leer los datos situados en las instrucciones DATA.

La instrucción READ es semejante a la instrucción INPUT, pero en lugar de leer del teclado, lee de las instrucciones DATA.

Después de la palabra READ aparecen los nombres de las variables separados por comas.

El mecanismo de lectura consiste en leer el dato que está preparado en la instrucción DATA. Cuando es la primera lectura, el dato es el primero de la instrucción DATA que tiene la numeración más baja.

El tipo del dato preparado y el de la variable a leer debe coincidir. En caso contrario se comete un error.

Si hacemos varias lecturas mediante la instrucción READ y no hay suficientes datos en las instrucciones DATA se comete un error.

En algunos BASIC es posible suprimir las comillas para los datos textuales. Esta manera de escribirlos no es recomendable pues se eliminan los blancos que hay detrás o delante del texto entre las dos comas que lo definen.

Una utilización de las instrucciones DATA es realizar búsquedas en tablas de datos fijos.

La búsqueda finaliza cuando se ha encontrado el dato buscado o bien cuando se ha encontrado una marca de final que nos indica que la búsqueda ha sido infructuosa.

Para poder realizar otra búsqueda se utiliza la instrucción RESTORE.

El efecto de esta instrucción es colocar como dato preparado para la lectura mediante el READ el primer dato de la instrucción DATA de más baja numeración.

Estas tres instrucciones son semejantes a una cinta magnética. Las instrucciones DATA definen el contenido de la cinta y el orden en que aparecerán los datos. La instrucción READ lee un dato y avanza la cinta hasta el próximo dato. La instrucción RESTORE rebobina la cinta al inicio.

EJERCICIOS AUTOCOMPROBACION

Complete las frases siguientes:

1. Los datos de un problema que varían muy poco en el tiempo se denominan ....

2. Para almacenar datos fijos en un programa se utiliza la instrucción ....

3. Los tipos de datos pueden .... en una instrucción DATA.

4. Los .... pueden colocarse en varias instrucciones DATA.

5. La .... de las instrucciones DATA puede ser cualquier parte del programa.

6. El orden de las instrucciones DATA define el .... de los datos.

7. Para acceder a un dato se utiliza la instrucción ....

8. La instrucción READ lee los valores de las variables que le siguen separadas por ....

9. Un dato no puede leerse si no se han .... los anteriores.

10. La instrucción que nos permite volver a iniciar la lectura de los DATA es la instrucción ....

Encierre en un círculo la letra que corresponda a la alternativa correcta.

11. La instrucción DATA en un programa se utiliza para almacenar:

a) Datos fijos.

b) Datos variables.

c) Funciones numéricas.

d) Dimensiones de las tablas.

12. Si tenemos 10 datos en un DATA que sucede al hacer el onceavo READ:

a) Repite el último leído.

b) Lee el primer dato otra vez.

c) Da un error.

d) Da cero si es numérico o vacío si es textual.

13. Cuando hacemos un READ de una variable numérica y el dato preparado en el DATA es textual ocurre que:

a) Se coloca cero.

b) Se salta los datos alfabéticos hasta encontrar uno numérico.

c) Toma el código ASCII del primer símbolo del texto.

d) Da un error.

14. Imagine que tiene 25 datos en un DATA, ha leído 14 con un READ, hace un RESTORE y finalmente otro READ. ¿Qué dato lee?

    a) El quince.

    b) El primero.

    c) Da error.

    d) El último.

15. Cada vez que se ejecuta una instrucción READ se lee:

    a) Un solo dato.

    b) Tres datos.

    c) Tantos como tenga la línea DATA.

    d) Tantos como variables tenga la instrucción READ.

16. Considere el programa siguiente:

```
10 DIM A(3)
20 FOR I = 1 TO 3
30 READ A(4-I)
40 NEXT I
50 FOR I = 1 TO 3
60 PRINT A(I)
70 NEXT I
80 DATA 1,2,3
```

¿qué secuencia de números se obtiene?

a) 1,2,3.

b) 3,2,1.

c) Da error.

d) 1,3,2.

17. Considere el programa siguiente:

```
10 READ A
20 IF A=0 THEN END
30 PRINT A
40 GO TO 10
50 DATA 5,4,0,2,3,1
```

¿qué secuencia de número escribe?

a) 5,4,0,2,3,1.
b) Da error.
c) 5,4.
d) 2,3,1.

18. Si al programa del ejercicio anterior se le cambia la instrucción 50 por:

```
50 DATA 5,4,3,2,1
```

¿qué secuencia de números escribe?

a) 5,4,2,3,1.
b) Da error.
c) 5,4.
d) 2,3,1.

19. Si en el programa del ejercicio 17 se cambia la línea 50 por:

```
50 DATA 5,4,"JUAN",0,2,3,1
```

¿qué secuencia de números escribe?

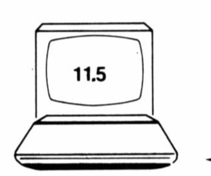
a) 5,4,0,2,3,1.
b) Da error.
c) 5,4.
d) 2,3,1.

20. Si en el programa del ejercicio 17 se cambia la línea 50 por:

```
50 DATA 3,2,0,"JUAN",5,6
```

¿qué secuencia de németros escribe?

a) 3,2,0,5,6.
b) Da error.
c) 3,2.
d) 5,6.



11.2 APLICACIONES

11.2.1 El mes en letras

Estas instrucciones, como vemos, tienen múltiples posibilidades de aplicación, ya que nos permiten almacenar datos dentro de un programa de una forma bastante cómoda.

Vamos a ver con otro ejemplo la utilización de estas nuevas instrucciones. Recuerde el programa que hemos visto en el capítulo anterior para convertir la fecha, escrita en formato:

DD-MM-AA

(por ejemplo 12-10-87, o 03-05-84) a la forma siguiente:

DD de mes de 19AA

(correspondería a 12 de octubre de 1987 para el primer caso, y a 3 de mayo de 1984 para el segundo).

Tal como ya habíamos dicho en el capítulo anterior, necesitaremos introducir la fecha en primer lugar, y separar la parte correspondiente al día (los dos primeros caracteres), el mes (los caracteres cuarto y quinto), y el año (dado por los dos últimos caracteres). Los símbolos de día y año se deberán escribir tal cual, y el mes se deberá traducir, determinando el nombre a partir del número del mismo. Finalmente, se escribirá la fecha completa.

La carga de una tabla fija

Ya hemos visto un programa, en el capítulo anterior, que nos permitía resolver este problema. Veamos cómo modifica el listado de utilización de las nuevas instrucciones vistas, DATA y READ.

```
NEW
 10 REM TRANSFORMACION DE LA FECHA
 20 REM Dimensionado de variables
 30     DIM N$(12)
 40 REM Lectura del nombre de los meses
 50     FOR I = 1 TO 12
 60         READ N$(I)
 70         NEXT I
 80 REM Entrada de la fecha en formato DD-MM-AA
 90     INPUT "Entre la fecha (DD-MM-AA): ",F$
100 REM Separacion y traduccion del mes
110     LET D$=LEFT$(F$,2)
120     LET M$=MID$(F$,4,2)
130     LET A$=RIGHT$(F$,2)
140 REM Traduccion del mes
150     LET M=VAL(M$)
160     LET M$=N$(M)
170 REM Construccion del resultado
180     LET R$ = D$+" de "+M$+" de 19"+A$
190     PRINT R$
1000 DATA "Enero", "Febrero", "Marzo"
```

```
1010 DATA "Abril", "Mayo", "Junio"
1020 DATA "Julio", "Agosto", "Septiembre"
1030 DATA "Octubre", "Noviembre", "Diciembre"
```

Ejecute el programa, e introduzca una fecha cualquiera, por ejemplo:

12-10-84

Obtendrá como resultado:

12 de Octubre de 1984

Analicemos con detalle cuál es el funcionamiento del programa. En primer lugar, leemos la fecha utilizando una variable textual, F$. Para conocer el día, el mes y el año, deberemos tomar por separado los valores correspondientes. Esta es la función de las instrucciones correspondientes a las líneas 110 a 130. A D$ se le asigna el día, a M$ el mes, y a A$ el año. El día y el año se escribirán tal cual, sin embargo, el mes debe traducirse.

Para cada mes, el número se corresponde a un nombre determinado. Así, el mes 1 es Enero, el 2 Febrero, etc., hasta el 12 que es Diciembre. Para ello, una vez separado el mes (en forma textual), tomamos el valor del mismo (línea 150). Por otra parte, y en las líneas 50 a 70 hacemos que el programa «conozca» la correspondencia entre cada número y los doce nombres asociados. Para ello utilizamos las instrucciones DATA y READ correspondientes, que asignan a la variable dimensionada N$, los nombres de cada mes.

Ahora, conociendo el número del mes que nos han introducido (valor que contiene la variable M), podremos saber el nombre del mismo, que vendrá dado por el valor de N$ cuyo subíndice corresponda al valor de M. Así, si M vale 1 deberemos tomar N$(1), que es Enero, si M vale 2, tomaremos N$(2), que será Febrero, etc. En general, tomaremos N$(M) y obtendremos el nombre del mes cuyo número viene dado por M (línea 160).

Una vez conocido esto, construiremos la variable resultado R$, a base de concatenar el número del día, la preposición «de», el nombre del mes, otra vez la preposión «de», y finalmente el número del año, precedido del 19. Por tanto, este programa, tal como está escrito, tendrá únicamente validez para este siglo. Si estuviéramos en el siglo XXI, deberíamos escribir un 20, no un 19.

¿Cómo podríamos modificar el programa para obtener la transformación de más de una fecha? Tal como ya hemos visto en casos anteriores, bastará añadir algunas líneas de forma que nos permitan finalizar o no, según se lo indique el usuario. Para ello escribiremos:

```
200    INPUT "Desea continuar (S/N): ", R$
210    IF R$="S" THEN GOTO 90
220 END
```

El programa completo quedará ahora:

```
NEW
 10 REM TRANSFORMACION DE LA FECHA
 20 REM Dimensionado de variables
 30    DIM N$(12)
 40 REM Lectura del nombre de los meses
 50    FOR I = 1 TO 12
 60       READ N$(I)
 70       NEXT I
 80 REM Entrada de la fecha en formato DD-MM-AA
 90    INPUT "Entre la fecha (DD-MM-AA): ",F$
100 REM Separacion y traduccion del mes
110    LET D$=LEFT$(F$,2)
120    LET M$=MID$(F$,3,2)
130    LET A$=RIGHT$(F$,2)
140 REM Traduccion del mes
150    LET M=VAL(M$)
160    LET M$=N$(M)
170 REM Construccion del resultado
180    LET R$ = D$+" de "+M$+" de 19"+A$
190    PRINT R$
200    INPUT "Desea continuar (S/N): ", R$
210    IF R$="S" THEN GOTO 90
220 END
1000 DATA "Enero", "Febrero", "Marzo"
1010 DATA "Abril", "Mayo", "Junio"
1020 DATA "Julio", "Agosto", "Septiembre"
1030 DATA "Octubre", "Noviembre", "Diciembre"
```

Ejecútelo para comprobar su funcionamiento. Podrá escribir las fechas que desee. En el momento que desee detener el proceso, bastará que lo indique cuando el programa le haga la pregunta, entrando una letra distinta de *S*.

Diferencias entre cargar una tabla y leer siempre en el DATA

¿Qué diferencia observa en este programa con respecto al anterior?

En este caso no utilizamos la instrucción RESTORE, y, a diferencia del caso anterior, no tenemos problemas con el almacén. Comparando los listados y el funcionamiento de uno y otro caso, se verá a qué se debe el distinto comportamiento de ambos.

En el caso del problema de traducción INGLES-ESPAÑOL, para cada palabra debemos recorrer el almacén, hasta encontrar la palabra o llegar al final del mismo, y cuando nos introducen una nueva, debemos recorrerlo otra vez, empezando por el principio. Por esta razón necesitamos la instrucción RESTORE. Sin embargo, en el programa de transformar la fecha, leemos los nombres de los meses una sola vez, y no es necesario que realicemos ninguna otra lectura. Por esto el bucle de lectura lo hemos si-

tuado al principio del programa, de forma que sólo se realice el proceso una sola vez.

11.2.2 El día en letras

Veamos ahora una posible ampliación de este programa. Supongamos que, además de traducir el mes, nos interese que el día aparezca escrito en letras. Por ejemplo, si la fecha es:

12-05-84

nuestro programa lo traduzca por:

Doce de Mayo de 1984

Dado que el programa ya nos traduce el mes, bastará añadir las instrucciones para traducir el día. Vamos a añadir las instrucciones para que, a partir del número de día —contenido en la variable D$—, nos de la correspondiente traducción en letras.

El programa deberá ser capaz de escribir en letras los números comprendidos entre uno y treinta y uno. Entre uno y veinte, todos los números son distintos. Deberemos almacenar pues, los nombres correspondientes a cada número.

A partir del número veinte y hasta el treinta, los nombres de los números se construyen con el prefijo «veinti» (correspondiente al 2) y la traducción del número que esté a la derecha de éste. Así, el número 25 se escribe «veinti» y a continuación «cinco»; será pues «veinticinco».

El treinta debe tenerse también almacenado, el 31 se construye a partir del anterior, añadiéndole una «Y», y la traducción del 1; será pues «treinta y uno».

Necesitaremos un almacén con los veinte primeros números, con el prefijo de «veinti» para construir los comprendidos entre 20 y 30, y con la traducción del 30. En total son 22 elementos. Estos se almacenarán en un DATA, que se leerá al principio del programa. La variable utilizada como almacén deberá dimensionarse.

La tabla de los nombres de los números. No es preciso almacenar el nombre de todos los números.

Por otra parte, tomaremos el valor del número, y haremos la traducción por separado, según sea inferior a 21, superior a 30, o esté entre ambos.

Al escribir el programa, ya tendremos en cuenta que se ha de unir al anterior; por tanto, utilizaremos nombres de variables que no puedan interferir, y numeraremos las instrucciones de forma conveniente, para que nos queden en el lugar adecuado. De esta forma no será necesario borrar el programa de traducción que ya tenemos escrito, y luego volver a escribirlo entero.

En primer lugar, deberemos escribir las instrucciones de lectura del DATA de los nombres de los números. Estas se escribirán a continuación de los nombres de los meses, la lectura deberá hacerse por tanto a continuación de la de aquéllos. La situaremos pues entre las líneas 70 y 80:

```
71 REM Lectura del nombre de los numeros
72   FOR I = 1 TO 22
73        READ P$(I)
74        NEXT I
```

Ahora, necesitamos efectuar la traducción del día. Esta se hará después de la del mes. Dado que necesitamos más de diez instrucciones, numeraremos las instrucciones entre 161 y 179, escribiendo de nuevo la instrucción REM que indica la construcción del resultado:

```
161 REM Traduccion del dia
162   LET V = VAL(D$)
163   IF V>20 THEN GOTO 166
164        LET D$=P$(V)
165        GOTO 180
166   LET R = VAL(RIGHT$(D$,1))
167   IF V>29 THEN GOTO 170
168        LET D$ = P$(21)+P$(R)
169        GOTO 180
170   LET D$ = P$(22)
171   IF R>0 THEN LET D$ = D$+" y "+P$(R)
175 REM Construccion del resultado
```

Por último, escribiremos los valores del DATA en el lugar correspondiente, que, como hemos dicho, será a continuación de los ya existentes:

```
2000 DATA "Uno","Dos","Tres","Cuatro","Cinco","Seis","Siete"
2010 DATA "Ocho","Nueve","Diez","Once","Doce","Trece"
2020 DATA "Catorce","Quince","Dieciseis","Diecisiete"
2030 DATA "Dieciocho", "Diecinueve", "Veinte"
2040 DATA "Veinti","Treinta"
```

Deberá modificarse la línea 30, que quedará:

```
30      DIM N$(12), P$(22)
```

El listado completo quedará pues:

```
 10 REM TRANSFORMACION DE LA FECHA
 20 REM Dimensionado de variables
 30    DIM N$(12), P$(22)
 40 REM Lectura del nombre de los meses
 50    FOR I = 1 TO 12
 60        READ N$(I)
 70        NEXT I
 71 REM Lectura del nombre de los numeros
 72    FOR I = 1 TO 22
 73        READ P$(I)
 74        NEXT I
 80 REM Entrada de la fecha en formato DD-MM-AA
 90    INPUT "Entre la fecha (DD-MM-AA): ",F$
100 REM Separacion y traduccion del mes
```

```
 110    LET D$=LEFT$(F$,2)
 120    LET M$=MID$(F$,3,2)
 130    LET A$=RIGHT$(F$,2)
 140 REM Traduccion del mes
 150    LET M = VAL(M$)
 160    LET M$ = N$(M)
 161 REM Traduccion del dia
 162   LET V = VAL(D$)
 163   IF V>20 THEN GOTO 166
 164       LET D$ = P$(V)
 165       GOTO 180
 166   LET R = VAL(RIGHT$(D$,1))
 167   IF V>29 THEN GOTO 170
 168       LET D$ = P$(21)+P$(R)
 169       GOTO 180
 170   LET D$ = P$(22)
 171   IF R>0 THEN LET D$ = D$+" y "+P$(R)
 175 REM Construccion del resultado
 180    LET R$ = D$+" de "+M$+" de 19"+A$
 190    PRINT R$
 200    INPUT "Desea continuar (S/N): ", R$
 210    IF R$="S" THEN GOTO 90
 220 END
1000 DATA "Enero", "Febrero", "Marzo"
1010 DATA "Abril", "Mayo", "Junio"
1020 DATA "Julio", "Agosto", "Septiembre"
1030 DATA "Octubre", "Noviembre", "Diciembre"
2000 DATA "Uno","Dos","Tres","Cuatro","Cinco","Seis","Siete"
2010 DATA "Ocho","Nueve","Diez","Once","Doce","Trece"
2020 DATA "Catorce","Quince","Dieciseis","Diecisiete"
2030 DATA "Dieciocho", "Diecinueve", "Veinte"
2040 DATA "Veinti","Treinta"
```

Escribamos RUN para comprobar el funcionamiento del programa. Las instrucciones añadidas para la lectura del nuevo DATA no precisan explicación, pues la lectura de los números se efectúa igual que la de los meses. Sí vamos a detenernos en cambio, en las instrucciones utilizadas para la traducción del día.

En primer lugar, tomamos el valor numérico del día introducido (línea 162). A continuación, comprobamos si el número es superior a veinte o no. Si no es superior utilizaremos el propio número como subíndice de la variable P$, que contiene los nombres correspondientes a cada número, y el valor obtenido lo almacenaremos en la variable D$ (línea 164) para que nos quede concatenado con el resultado.

Si el número es superior a 20, separaremos la parte derecha del mismo (línea 166), y tomaremos su valor, que asignaremos a la variable R. El proceso es ahora distinto si estamos por debajo o por encima de treinta. Si es inferior a treinta, concatenaremos el prefijo correspondiente (que ocupa la posición 21 de nuestra tabla de datos) con la traducción de la parte derecha del número. Esta vendrá dada por el elemento de la variable P$ que ocupe la posición dada por el valor R.

Esto es así, porque este valor de R (correspondiente a las unidades), estará comprendido entre 1 y 9, y como tal lo tendremos almacenado en la tabla.

Finalmente, queda el caso de los números superiores a 29. En primer lugar, asignaremos la primera parte del resultado, que será el último elemento de nuestra tabla. Si el valor era 30, ya está hecha la traducción. Si por el contrario, era superior, tal como se controla en la línea 171, habrá

que añadir la conjunción «y», así como la traducción de las unidades, cuyo valor se obtiene como en el caso anterior.

Dado que el número máximo que nos pueden introducir es el 31 (no existen meses de más de 31 días), se hubiera podido construir esta última parte de una forma más simple. Sin embargo, tal como está escrito es válido para un caso general. A partir de éste, piense cómo se podría construir un programa para traducir números mayores, en principio inferiores a 100.

11.2.3 La agenda

Veamos por último un nuevo ejemplo de utilización de estas instrucciones. Vamos a construir un programa que nos permita la manipulación de una agenda. En ella tendremos, para cada persona, el nombre, la dirección y el número de teléfono, y podremos conocer todos los datos de cada uno entrando uno cualquiera de estos tres. Para ello, dispondremos los datos en forma de tabla con dos subíndices, con la siguiente estructura:

	1	2	3
	NOMBRE	DIRECCION	TELEFONO
1	Fernández Roca, Javier	C/. Pino, 32	346-98-80
2	García Llopart, Elena	C/. Agua, 12	256-98-73
3	Merlo Rubio, David	C/. Arenas, 23	423-90-65
"	"	"	"
"	"	"	"
"	"	"	"

De esta forma, el elemento 1,1 de la tabla será el primer nombre, el 2,2 será la calle del segundo elemento, el 1,3 será el teléfono del primero, etc.

Organización de un conjunto dimensionado para la consulta

Así, el primer subíndice indica el orden en la lista (lugar ocupado en la misma), mientras que el segundo indica si se trata del nombre, de la dirección o del teléfono. Evidentemente esta lista se puede hacer más larga, ya sea por abajo, aumentando el número de personas que la componen, tal como se indica en la tabla anterior, o bien lateralmente, indicando más características de cada una de estas personas. Así, para cada uno podríamos tener el número de afiliación a la Seguridad Social, el título u oficio que posee, etc. Estos campos nos alargarían la lista por la derecha.

El programa de consulta de esta agenda, se basará en esta estructura. Constará de una primera fase de lectura de la agenda, y asignación a la correspondiente variable textual subindicada. Una vez «conocidos» por el programa todos los elementos que integran la agenda, realizaremos el proceso de consulta. En primer lugar, el usuario de nuestro programa nos deberá indicar si quiere realizar la consulta de un nombre, de una calle, o de un teléfono. Una vez indicado esto, deberá introducirnos el texto a buscar, que debe corresponder a lo solicitado. Así, si se ha indicado que se desean

conocer los datos de una persona de la que únicamente se sabe el número de teléfono, se indicará que se va a consultar un teléfono, y a continuación se escribirá el número a buscar.

Si el número buscado existe en la lista, se indicarán todos los datos del propietario del mismo (en este caso nombre y dirección), y se seguirá el recorrido, hasta el final de la lista. Debe permitirse la realización de tantas consultas como se desee.

El listado para ello podría ser:

```
NEW
 10 REM AGENDA
 20 REM Dimensionado e inicio de la tabla
 30    DIM A$(50,3) : LET I=0
 40 REM Lectura de los elementos de la agenda
 50     READ M$, N$, L$
 60     IF M$="FIN" THEN GOTO 120
 70        LET I = I+1
 80        LET A$(I,1)=M$
 90        LET A$(I,2)=N$
100        LET A$(I,3)=L$
110        GOTO 50
120 REM Presentacion y entrada de datos
130     LET NE = I
140     CLS : PRINT TAB(15);"AGENDA"
150     PRINT : PRINT : PRINT : PRINT
160     INPUT "Entrada por nombre (N), direccion (D),
    telefono (T) o Fin (F): ",R$
170     IF R$<>"N" THEN GOTO 200
180        LET J = 1
190        GOTO 280
200     IF R$<>"D" THEN GOTO 230
210        LET J = 2
220        GOTO 280
230     IF R$<>"T" THEN GOTO 260
240        LET J = 3
250        GOTO 280
260     IF R$<>"F" THEN GOTO 140
270        GOTO 420
280 REM Entrada del dato solicitado
290     INPUT "ENTRE EL VALOR: ",D$
300 REM
310 REM Busqueda a lo largo de la tabla
320     CLS : LET I=1 : LET T=0
330     IF I>NE THEN GOTO 390
340        IF D$<>A$(I,J) THEN GOTO 370
350           LET T = T+1
360           PRINT A$(I,1);" ";A$(I,2);" ";A$(I,3)
370        LET I = I+1
380        GOTO 330
390     PRINT "Elementos que cumplen la condicion : ";T
400     IF INKEY$="" THEN GOTO 400
410     GOTO 140
420 END
1000 DATA "Fernandez Roca, Javier","C/. Pino, 32","346-98-80"
1010 DATA "Garcia Llopart, Elena","C/. Agua, 12","256-98-73"
1020 DATA "Merlo Rubio, David","C/. Arenas, 23","423-90-65"
1030 DATA "FIN","",""
```

Comprobemos el funcionamiento del programa escribiendo RUN. Supongamos que deseamos saber cuántos elementos existen en la agenda

que vivan en la calle del Pino, 32. Para ello, pulsaremos una D, indicando que queremos realizar la consulta de una dirección, y a continuación escribiremos:

C/. Pino, 32

El programa repasará la tabla. Dado que hay un elemento en la misma que cumple la condición pedida, escribirá todos sus datos en la pantalla:

Fernández Roca, Javier C/. Pino, 32 346-98-80

A continuación escribirá:

Elementos que cumplen la condición dada: 1

y esperará que pulsemos una tecla para continuar, que en este caso será volver al principio del programa para permitirnos realizar una nueva consulta.

Ponga atención al escribir el texto que busca. Asegúrese de que está escrito *exactamente igual* que en el almacén —mayúsculas y minúsculas en la misma posición, espacios en blanco, comas, etc.—, pues de lo contrario no lo podrá encontrar al hacer la búsqueda.

Los datos de la agenda están en una tabla DATA

Veamos con detalle cuál es el funcionamiento del programa. En primer lugar, se realiza la lectura de la tabla. Esta tabla se asigna a una variable textual, que previamente habremos dimensionado (línea 30). En principio, no conocemos cuántos elementos constituyen nuestra agenda; por esta razón, se dimensiona la variable correspondiente a un valor alto, que en principio parezca bastante superior al número de elementos que podamos tener (en este caso hemos dimensionado a 50, si parece poco puede dimensionarse a 100 o más).

En cuanto al número de datos para cada elemento, hemos supuesto que en este caso tenemos tres: nombre, dirección y teléfono. Si se precisan más datos, bastará aumentar el segundo subíndice al dimensionar la variable utilizada para contener la tabla.

El proceso de lectura se realiza entre las líneas 50 y 110. Dado que no conocemos cuántos elementos tenemos, ya que estos pueden además sufrir variaciones (se pueden añadir o borrar elementos), deberemos indicar el final del almacén mediante una señal determinada. En este caso, escribiremos la palabra FIN.

En primer lugar, leeremos tres valores, sin asignarlos aún a la variable tabla, sino a tres variables auxiliares cualquiera. Comprobaremos entonces que el primer valor leído no sea la marca de final. Si efectivamente no es así, los valores leídos forman parte de la tabla, deberán constituir pues el siguiente elemento de la misma. Incrementaremos entonces el subíndice correspondiente (I), y asignaremos cada dato en la posición correspondiente (primer valor para la primera columna, segundo para la segunda...) Esto se gobierna mediante el subíndice correspondiente (el de columna: J).

Este proceso se repite hasta que se lee la marca de final: FIN, momento en que se da por terminada la lectura, y se pasa a la fase siguiente.

Por otra parte, dado que en cada ocasión realizamos la lectura de tres variables a la vez, la última instrucción DATA, que marca el final del almacén, deberá contener tres elementos. El primero es el que nos sirve de

control, y contiene la palabra FIN. Los dos restantes pueden ser nulos, los incluiremos después de esta palabra.

Dado que ahora la variable I contiene el número de elementos que existen en la tabla, almacenaremos este valor para su posterior utilización a lo largo del programa (línea 130). A continuación se realiza la presentación de pantalla (líneas 140 y 150), y la pregunta de lo que se desea entrar (línea 160). Esta pregunta nos indicará qué columna se ha de recorrer: la de nombres (pulsando una «N»), de direcciones (pulsando una «D») o de teléfonos (pulsando una «T»). Según lo que se desee, el subíndice de columna valdrá 1, 2 ó 3.

Simultáneamente, en este punto se puede indicar que se desea finalizar el proceso, para lo que basta pulsar una «F». Si se escribe alguna otra cosa por error, el programa repite de nuevo la pregunta.

La búsqueda en la agenda

La fase siguiente es la de entrada del valor a buscar (línea 290), y finalmente la realización de la búsqueda efectiva. Esta consiste, como ya hemos dicho, en recorrer la columna de la agenda que nos hayan indicado, comparando cada uno de sus elementos con el que nos han entrado. La búsqueda se empieza por el primer elemento; en la misma línea (320) se inicializa un contador para contabilizar los elementos que cumplan la condición solicitada. La búsqueda finaliza cuando se ha recorrido toda la tabla. Cuando se encuentra un elemento que cumple la condición de igualdad con el pedido, se presentan todos los datos del mismo: nombre, dirección y teléfono, y se prosigue el recorrido de la tabla, hasta el final de la misma.

Por último, se indica el total de elementos que cumplen la condición pedida (línea 390), se espera que el usuario haya leído el resultado (línea 400) y se vuelve de nuevo al principio del programa.

Observe que en este caso sólo realizamos la lectura del almacén una sola vez. Este mismo sistema podríamos haberlo utilizado en el programa traductor, leyendo todo el diccionario al principio de la ejecución, y asignándolo a una variable tabla como en este caso. De esta forma no hubiéramos necesitado la instrucción RESTORE. A efectos prácticos, no observaríamos diferencia alguna en el funcionamiento entre una u otra forma, aunque, desde el punto de vista lógico, parece más correcto efectuar una sola vez al principio la lectura de todo el almacén, asignándolo a una variable determinada, sobre la cual realizar el proceso de búsqueda.



11.3 FUNCIONES DEFINIDAS POR EL USUARIO

En el capítulo 3 del primer volumen estudiamos las funciones intrínsecas del BASIC, por ejemplo SQR(X), LEN(X$), INT(X), etc.

Recordemos, que estas funciones las incorpora internamente el ordenador. Todas ellas están predefinidas y su funcionamiento preestablecido. Algunas variantes del BASIC, permiten además, que el usuario se defina sus propias funciones.

Estas funciones se usan en el programa de forma idéntica a como se utilizan las funciones intrínsecas. No obstante estas funciones definidas por el programador no perduran más allá del programa en que se han definido.

Si hay que usarlas en otros programas es necesario volverlas a definir en cada uno de ellos.

Normalmente se utilizan funciones definidas cuando es necesario realizar frecuentemente a lo largo del programa unas operaciones iguales pero para valores diferentes. Es más interesante aún cuando estas operaciones son relativamente complicadas. Algunos ejemplos son las funciones de cálculo del interés compuesto, cálculo de superficies y volúmenes de cuerpos sólidos.

Los ejemplos más claros son precisamente las funciones intrínsecas que nos permiten disponer de resultados en operaciones matemáticas complejas, por ejemplo, sacar logaritmos, o no tan complejas como es sacar el valor absoluto.

Se requieren dos tipos de instrucciones

Estas funciones definidas por el programador requieren dos tipos de instrucciones:

Uno, en que se definen las operaciones a realizar.

Otro, en el que utilizamos las operaciones definidas por el cálculo de algún valor.

El problema nuevo que afrontamos aquí es la definición de las funciones, ya que la utilización es idéntica, salvo el convenio de nombres, al de las funciones intrínsecas.

Antes de abordar el problema de la definición estudiaremos la nomenclatura de estas funciones.

11.3.1 Nomenclatura

Las funciones intrínsecas, que ya conocemos, todas tienen un nombre, por ejemplo, ASC(X$), LOG(X), VAL(X$), etc. Pues bien, lo primero que tenemos que ver es el nombre que hay que dar a la función, para que el ordenador sepa que se trata de una función definida por el usuario.

El nombre de la función debe comenzar obligatoriamente por las letras FN, que provienen de la abreviatura de FUNCION (FUNCTION en inglés).

Después del prefijo FN, se coloca la letra que identifica la función. Por ejemplo:

FNA   FNX

Son nombres correctos de funciones definidas por el usuario.

Si la función tiene un resultado textual, después del nombre hay que añadir el símbolo de dólar. Por ejemplo:

FNB$ FNZ$

Son funciones definidas por el usuario con un resultado textual.

El hecho de que después del prefijo FN podamos poner letras distintas, permite el que en un programa podamos definir más de una función. Sin embargo, lo que no podemos hacer es tener más de una función definida con la misma letra, pues aun cuando el BASIC no diera error esta ambigüedad sería inadmisible desde un punto de vista lógico.

Aunque hay versiones de BASIC que permiten que después del prefijo FN pongamos más de una letra, nosotros seguiremos la norma de poner siempre una sola letra para hacer compatibles nuestros programas con las distintas versiones de BASIC.

Tenga en cuenta también, que al principio puede confundirlas con los conjuntos dimensionados debido a que van seguidas de paréntesis.

Por ejemplo, si usted encuentra en un programa

FNS(M)

le podría parecer que es una variable de un conjunto dimensionado, en aquellas versiones de BASIC que admiten más de una letra para las variables de conjuntos dimensionados. Sin embargo, esta confusión no se le dará, pues aun en el caso de que el BASIC admita variables de conjuntos dimensionados con más de una letra, nunca podrán empezar estas variables por el prefijo FN. Téngalo usted también en cuenta para no utilizar variables que comiencen por estas letras.

Por tanto, el nombre de una función definida por el usuario debe comenzar siempre por el prefijo FN seguido de una letra, cuando el resultado es numérico, y seguido de una letra y el símbolo de dólar, cuando el resultado es textual.

Según esto, serían nombres incorrectos de estas funciones los siguientes:

FAN (Pues debe empezar por el prefijo FN)

FN1 (Pues al prefijo FN debe seguir una letra)

FN$ (Pues falta una letra entre el prefijo FN y el $)

11.3.2 Definición de las funciones

En el apartado anterior hemos visto qué nombre hay que dar a las funciones definidas por el usuario. Ahora veremos el modo de definir estas funciones.

El proceso de definición de las funciones consta de tres partes:

a) Asignar el nombre a la función a definir.

b) Especificar los argumentos.

c) Especificar las operaciones que debe realizar.

Vea ahora la figura 1, donde se identifican cada una de estas partes en un ejemplo concreto, y téngala presente a medida que vaya estudiando lo que sigue.

Asignación del nombre

a) *Asignación del nombre*. La instrucción que nos permite definir las funciones es la instrucción DEF, que proviene de DEFINIR (DEFINE en inglés).

Detrás de la palabra DEF, que es una de las palabras reservadas por el BASIC, se coloca el nombre de la función a definir. En el ejemplo de la

```
DEF        FNF      (I , R , N)   =   I * (1 + R)^N
```

Asignación del nombre	Especificación de los argumentos	Especificación de las operaciones

Figura 1. Definición de una función propia

figura 1, este nombre será concretamente FNF. Observe que en el nombre de la función se cumple lo que hemos dicho sobre la nomenclatura de estas funciones. Es decir, que se coloca el prefijo FN seguido de una letra. En este caso una F.

Especificación de los argumentos

b) *Especificación de los argumentos.* Después del nombre se abre un paréntesis y a continuación se especifica el nombre o nombres de los argumentos, separados por comas si hay más de uno y finalmente se cierran los paréntesis.

Veamos ahora con un poco más de detención el concepto de argumento, para analizar cuáles son las características de estos nombres que colocamos entre paréntesis. Para ello, tomaremos como ejemplo la fórmula del interés compuesto, que es la siguiente:

$$F = I * (1 + R) \hat{} N$$

donde:

F = Valor final

I = Valor inicial

R = Rédito

N = Número de años.

Esta forma de expresar la fórmula es matemática, pero está claro que obtenemos un número *F*, a partir de los tres números *I, R* y *N*. Estos tres números *I, R* y *N* son precisamente los argumentos. El argumento es, pues un nombre, que indica que para calcular es preciso conocer el valor de ese nombre. En el ejemplo concreto de esta fórmula será necesario conocer el valor numérico de cada uno de estos argumentos.

En el ejemplo de la fórmula, el valor es numérico, y por eso el nombre de cada uno de ellos está representado por una letra. Sin embargo, el argumento o argumentos podrían ser también textuales. En este caso, después de la letra que indica el nombre del argumento deberíamos colocar el símbolo dólar.

Especificación de las operaciones

c) *Especificación de las operaciones.* Sigamos ahora con el ejemplo de la figura 1. Una vez que hemos especificado el nombre y la lista de los argumentos vemos en la figura que se coloca el signo igual y a continuación

se especifican las operaciones que van a intervenir en el cálculo de la función.

La manera de especificar estas operaciones es exactamente como una expresión normal de BASIC. Por tanto, se podrán especificar todas las operaciones que tengan sentido en BASIC.

En la fórmula concreta del interés compuesto que hemos puesto como ejemplo vemos que aparecen mezclados los nombres de los argumentos I, R y N con los operadores (*, + y ^) y con constantes (en este caso sólo el 1). Pero esta mezcla es una expresión perfectamente válida en BASIC.

En la expresión intervienen los nombres de los argumentos, de modo que en el momento de cálculo, el BASIC sustituye dichos argumentos por los valores numéricos que se especifican en el momento de la llamada.

Formato general de la instrucción DEF

En conclusión, el formato general de la instrucción DEF es el siguiente:

DEF FN <nombre> (<arg. 1>, <arg. 2>,...) = <expresión>

en donde:

nombre es por lo menos una letra escogida por el programador. Se le añade el símbolo dólar si el resultado ha de ser textual.

argumento es un nombre que nos indica que al llamar a la función es necesario dar este valor. Este nombre se denomina argumento. Si el nombre termina con el símbolo dólar, significa que el argumento es textual. No hay ningún problema que haya argumentos textuales y numéricos en una misma instrucción DEF, siempre que su utilización en la expresión sea coherente en BASIC, o incluso, que se mezclen argumentos textuales y numéricos.

expresión es el conjunto de operaciones que deben realizarse para calcular el valor de la función y que, insistimos, deben ser operaciones admitidas en BASIC.

Por otra parte, las características que debe recordar al utilizar la instrucción DEF son las siguientes:

a) Tiene una cierta similitud con una instrucción LET. La diferencia fundamental está en que los cálculos, en la instrucción DEF, no se efectúan inmediatamente, sino que se difiere hasta que se llama a la función dentro de una instrucción de BASIC, tal como PRINT o LET. Esto es además lógico, pues es al llamar a la función cuando se dan los valores de los argumentos. Por ejemplo, en el caso que hemos comentado de la fórmula del interés compuesto y suponiendo que el valor inicial es de 100000, el rédito del 10 y los años 5, obtendríamos el valor final (F) mediante la siguiente instrucción:

```
PRINT FNF(100000,0.1,5)
```

b) La posición de esta instrucción dentro del programa (es una sentencia numerada igual que las demás) varía según la versión de BASIC. A

veces se requiere que la función esté definida antes de que sea utilizada por primera vez. En otros casos la instrucción DEF puede situarse en cualquier parte del programa.

c) Los argumentos en la instrucción DEF no tienen nada que ver con las variables del BASIC. Es decir, estos nombres sólo se utilizan internamente en la instrucción DEF para relacionar los cálculos y los datos que le suministramos. Cuando utilizamos la función del interés compuesto, si tenemos variables con el nombre I, R o N en el programa, no ocurrirá ninguna confusión; las variables son entidades totalmente separadas de los argumentos en la definición de las funciones.

d) Los nombres de los argumentos pueden repetirse en distintas definiciones de funciones sin que se interfieran unos a otros. Recuerde que son sólo indicadores para saber cómo deben sustituirse en la expresión los valores de los argumentos.

11.3.3 Ejemplos con la instrucción DEF

Para finalizar con la instrucción DEF veamos algunos ejemplos de definición de funciones.

a) Fórmula del interés compuesto. Ya la hemos analizado arriba, su valor es

```
DEF FNF ( I,R,N) = I*(1+R)^N
```

b) Cálculo del volumen de un depósito cilíndrico.

El volumen de un depósito cilíndrico depende del radio de la base y de la altura mediante la fórmula:

$$V = \pi * R * R * H$$

en donde, R es el radio de la base y H la altura. π es la constante PI que vale 3.1416.

La definición de la función es

```
DEF FNV (R,H) = 3.1416*R*R*H
```

Se trata de una función de nombre FNV, por lo tanto, de resultado numérico con dos argumentos ambos numéricos, R y H. Las operaciones son simples multiplicaciones entre la constante π y los argumentos numéricos.

c) Conociendo tres valores numéricos de día, mes y año escribir un texto que nos los dé, separados por guiones, tal como se suelen escribir las fechas.

Las diferencia de esta función con las anteriores es que el resultado es un texto y hace que sea más difícil expresarla en forma de fórmula, pues las matemáticas suelen trabajar con números.

A partir del enunciado es claro que tiene tres argumentos numéricos: día, mes y año. Las operaciones son pasar a texto estos números y unirlos mediante guiones para escribir las fechas según la costumbre. Esto que hemos descrito lo plasmamos en la instrucción DEF del modo siguiente:

```
DEF FND$ (D,M,A) = STR$(D)+"-"+STR$(M)+"-"+STR$(A)
```

Observe que el nombre de la función es D seguida del símbolo dólar, pues el resultado es textual. El resto de la instrucción refleja exactamente la descripción de las operaciones. Se utiliza la función intrínseca STR$ en la definición de las operaciones. Ya hemos dicho que es válida cualquier expresión en BASIC.

d) La función nos da el número de blancos que hay que añadir a una cadena de caracteres para que se ajuste por la derecha en una línea de una longitud determinada.

También en este caso, es difícil encontrar una fórmula que especifique este enunciado claramente. De todas maneras, la función debe tener un resultado numérico ya que se desea el número de blancos a añadir. Tiene además dos argumentos, unó textual y otro numérico, éste nos indica en qué columna hay que ajustar el texto.

Las operaciones a realizar son restar del ancho total la longitud que tiene el texto de tal manera que nos dará el número de blancos que hay que añadir a la izquierda para que el texto quede ajustado a la derecha.

Por ejemplo. Si quiero colocar la palabra *Atentamente* ajustada a la derecha de una carta de 60 columnas. Para hacer esta impresión debo añadir 60 blancos menos la longitud de la palabra *Atentamente*; es decir, 11, lo que da un total de 49 espacios en blanco.

La instrucción DEF a colocar es

```
DEF FNC( T$,M) = M - LEN(T$)
```

en donde, T$ es el argumento textual y M el argumento que indica el margen al cual queremos ajustar.

11.3.4 Utilización de las funciones

Para ilustrar la utilización de las funciones definidas tomaremos el ejemplo del cálculo del volumen de un depósito cilíndrico, pero ahora las presentaremos ya numeradas formando parte del programa.

En primer lugar escribimos la línea 10 en la que definiremos la función FNV, cuya misión será la de calcular el volumen de un depósito cilíndrico. Tenemos por tanto:

```
10 DEF FNV(R,H) = 3.1416 * R * R * H
```

Ahora ya sólo tenemos que llamar a la variable definida mediante un PRINT, por ejemplo, para que realice el cálculo, sin olvidar que al llamar a la función debemos darle los valores de los argumentos que llevaba en la definición. En nuestro ejemplo, vamos a suponer que el radio (R) es de uno y la altura (H) es de dos. Escribiríamos por tanto:

```
20 PRINT FNV(1,2)
```

Al escribir RUN nos dará un resultado de 6.2832.

Veamos ahora diversos ejemplos, unos con resultados numéricos y otros con resultados textuales, unos con argumentos numéricos y otros con argumentos textuales, e incluso mezclando los argumentos numéricos y los textuales. Iremos numerando las líneas para cada uno de los ejemplos:

a) Función con resultado numérico y argumentos también numéricos:

```
10 DEF FNF(I,R,N) = I * (1+R)^N
```

b) Función con resultado textual y argumentos numéricos:

```
20 DEF FND$ (D,M,A)= STR$(D)+"-"+STR$(M)+"-"+STR$(A)
```

c) Función con resultado numérico y argumentos textuales:

```
30 DEF FNJ(A$,B$) = VAL(A$)*VAL(B$)
```

d) Función con resultado textual y argumentos textuales:

```
40 DEF FNH$(X$,Y$) = X$+Y$
```

e) Función con resultado numérico y argumentos textual y numérico:

```
50 DEF FNC (T$,M) = M- LEN(T$)
```

f) Función con resultado textual y argumentos textual y numéricos:

```
60 DEF FNK$(H$,P) = STR$(P) + H$
```

Como puede ver se pueden hacer múltiples combinaciones, a condición de que a la hora de hacer la especificación de las operaciones el resultado tenga sentido en BASIC.

Ahora daremos valores a los argumentos en las líneas que siguen, haciendo que se escriban los resultados mediante la instrucción PRINT. Agruparemos los resultados de dos en dos, siguiendo el mismo orden en que hemos escrito las definiciones de las funciones. Así tendremos:

```
70 PRINT FNF(100,0.1,3), FND$(1,1,87)
80 PRINT FNJ("0025","002"), FNH$("SACA","CORCHOS")
90 PRINT FNC("ATENTAMENTE",60),FNK$("M",3)
```

Al escribir RUN veremos que nos dará el resultado de

133.1	1-1-87
50	SACACORCHOS
49	3M

Hagamos ahora un ejemplo algo más complicado. Se trata de construir una tabla que nos dé el valor final de 100 unidades monetarias, con réditos alternativos que van desde el 8 al 14 % y desde 3 a 5 años.

El programa consta de dos bucles: El primero que recorre los réditos y el segundo los años. En el interior del bucle se escribe el valor de la función del interés compuesto.

El programa sería el siguiente:

```
10 DEF FNF(I,R,N) = I*(1+R)^N
50 FOR I = 0.08 TO 0.14 STEP 0.01
60 FOR N = 3 TO 5
70 PRINT "R=";I;" N=";N;" Valor=";FNF(100,I,N)
80 NEXT N
90 NEXT I
```

observe que se inicia en la línea 10 con la definición de la función.

Ejecute el programa. Obtiene ahora una tabla que le va dando el valor final para distintos valores del rédito y los años.

El punto más importante a notar no son los resultados sino la manera como se ha utilizado la función. Observe que se han utilizado las variables I y N que son argumentos. Incluso la variable I no se corresponde con el argumento primero. Debe recordar que los argumentos y los nombres de las variables no tienen nada que ver.

Características de la utilización de las funciones

Las características de utilización más importantes de las funciones son:

1. Se pueden utilizar en el lugar donde se pueda colocar una variable o una constante del mismo tipo que el resultado de la función. Excepto en la parte izquierda de la asignación. No tiene sentido una instrucción como

```
LET FNV(1,2) = A* 5+B
```

sí en cambio, puede utilizarse en expresiones tan complicadas como

```
LET A = FNV(1,2)*3+B-SQR(7)
```

2. Los argumentos deben coincidir en la definición y en la utilización tanto en tipo, como en número, como en orden.

Es decir, cada vez que utilizamos la función FND$ debemos suministrar tres argumentos numéricos, el primero debe ser el día, el segundo el mes y el tercero el año.

Cuando utilizamos la función FNC se debe suministrar dos argumentos el primero de tipo textual y el segundo de tipo numérico.

Cualquier discordancia en el número de argumentos, el tipo o el orden, el BASIC la señala como error.

Tenga en cuenta que el BASIC sólo distingue entre números y textos; en el ejemplo que hemos mencionado de la función FND$ Ud debe suministrar tres argumentos numéricos, lo hace pero se equivoca y da primero el año, después el mes y en tercer lugar el día. El BASIC no le da error porque no distingue si un número representa un año o un día. En cambio en el segundo ejemplo el de la función FNC, sí que obtendremos error si damos primero el dato de la columna y después el texto. El BASIC distingue claramente entre textos y números.

3. En el momento de la utilización cada argumento puede suministrarse en forma de una expresión válida en BASIC y del nivel de complejidad que se quiera.

En el programa que hemos mostrado de ejemplo ya hemos visto cómo se pueden utilizar variables como argumentos. Recuerde la sentencia

```
70 PRINT "R=";I;" N=";N;" Valor=";FNF(100,I,N)
```

También se pueden utilizar expresiones más complicadas. Por ejemplo, la instrucción

```
PRINT FNV( 5+2*I*(J-4),2/SQR(7+L) )
```

es válida en el supuesto que estén definidas las variables I, J y L. Como observa, cada argumento es una expresión de una complejidad más o menos grande.

Para terminar vamos a considerar un caso de definición de una función que hemos olvidado hasta este momento para que las ideas queden más claras.

En la definición de una función puede utilizarse una variable del programa en la definición de las operaciones

Considere la definición siguiente:

```
50 DEF FNX (X,Y) = D*D/4 - X*X - Y*Y
```

Se trata, en principio, de una definición normal. En primer lugar el nombre es FNX, correcto según las reglas de nomenclatura que se han estudiado. A continuación vienen los argumentos que son el X y el Y separados por comas y entre paréntesis. El símbolo igual y finalmente la expresión de las operaciones. La única anomalía a notar es precisamente en la definición de las operaciones la aparición del nombre D. No es un argumento porque no está en la lista. ¿De qué se trata entonces?, la respuesta es sencilla es una variable de BASIC. Es decir, cuando definimos una función todos los nombres que utilicemos que no estén en la lista de argumentos se considerarán variables de BASIC. En el ejemplo concreto la D debe responder a una variable en BASIC.

¿Qué ocurre cuando utilizamos la función? Pues en este caso toma el valor de la variable en el momento que se está ejecutando.

Veamos el ejemplo del siguiente programa: (se añade la línea 50 que define la función y que hemos dado más arriba).

```
60 LET D= 10
70 PRINT FNX(3,4)
80 LET D= 20
90 PRINT FNX(3,4)
```

Al ejecutarlo obtenemos en el PRINT de la línea 70 un 0 y en el de la línea 90 un 75.

La diferencia de valores proviene de que hemos alterado el valor de D en la línea 80. En la línea 70 el valor de D es 10 en el cálculo de la función. En la línea 90 el valor de D es 20 en el cálculo de la función.

Este mismo ejemplo nos demuestra que no es nada aconsejable utilizar este tipo de recursos; la misma función llamada con los mismos argumentos en distintas partes del programa da resultados diferentes.

De todas maneras es posible utilizar una variable del BASIC en el interior de la definición de las funciones aunque, en general, no es una técnica aconsejable.

El BASIC considera variables en una instrucción DEF a todos los nombres que no aparecen en la lista de argumentos.

RESUMEN

La búsqueda en listas de datos fijos es una necesidad muy frecuente en los programas.

Los datos fijos se almacenan en instrucciones DATA.

Se han estudiado tres técnicas para manipular estos datos fijos:

1. Con la instrucción READ y RESTORE.
Las características de este tipo de búsqueda son:

a) Cada vez que se realiza la búsqueda se lee el DATA hasta encontrar el elemento buscado o hasta alcanzar un elemento que nos marca el final.

b) Para iniciar otra búsqueda utilizamos el RESTORE.

Este esquema es el más económico en memoria pero el más incómodo de programar, siempre hay que acceder a un elemento detrás de otro.

El ejemplo del programa traductor utiliza esta técnica.

2. Colocar los elementos en una tabla adicional de la dimensión justa.

Consiste en leer los datos al inicio del programa en un conjunto dimensionado. Se parte del supuesto que conocemos el tamaño de la tabla.

Se utiliza en el ejemplo de almacenar los nombres de los meses.

De hecho, duplicamos en memoria la lista de los meses, la contenida en el DATA y la de la tabla. Sin embargo, el acceso a los datos por número es una comodidad importante.

3. Utilizar una tabla de dimensiones mayores que las que precisamos y leemos del DATA hasta encontrar un marcador de final.

Este método es muy parecido al anterior en cuanto a economía de memoria.

Tiene la ventaja respecto al anterior que la modificación de instrucciones DATA no altera en nada el programa. Es el ejemplo de la agenda.

El BASIC permite definir funciones por parte del programa que sólo tienen validez dentro del ámbito del programa y se utilizan de una manera muy parecida a las funciones intrínsecas estudiadas en el capítulo 3 del primer volumen de la Enciclopedia.

El BASIC obliga a que estas funciones empiecen por las letras FN para distinguirlas de los conjuntos dimensionados. Como contrapartida los conjuntos dimensionados no pueden empezar con las letras FN.

Le sigue una o más letras que elige el programador y si el resultado es textual debe finalizar con un dólar ($).

A continuación aparecen encerrados entre paréntesis una lista de variables separadas por comas que son los argumentos de la función. De la misma manera que utilizamos las funciones LEN, MID, SQR, etc.

Las variables que sirven de argumentos pueden ser numéricas o textuales.

Para utilizar estas funciones es necesario definirlas.

Se utiliza la instrucción DEF para realizar esta definición. Cuando

definimos una función con la instrucción DEF sólo la definimos y no se calcula nada.

En el proceso de definición especificamos:

a) El nombre de la función.

b) El número y tipo de argumentos que utiliza la función.

c) Las operaciones que realizaremos con los argumentos para alcanzar el resultado deseado.

Los argumentos en la definición no son variables del programa. Cuando se calcule la función serán reemplazados por las variables que utiliza la función en el momento de la llamada.

Los argumentos en la utilización deben coincidir en número, tipo y orden respecto a los argumentos utilizados en la definición. En caso contrario el BASIC da error.

Las operaciones que se pueden utilizar en la definición son cualquiera de los operadores, funciones intrínsecas o funciones propias que son correctas en BASIC.

En cuanto a las variables se pueden utilizar cualquiera del programa además de los argumentos definidos en la lista.

En caso de conflicto siempre prevalece el argumento.

EJERCICIOS AUTOCOMPROBACION

Complete las frases siguientes:

21. Para cargar una tabla almacenada en instrucciones DATA puede utilizarse un dato que nos indique el .... de la tabla.

22. El programador puede definir .... propias en el BASIC.

23. Las funciones propias en BASIC empiezan por las letras ....

24. Si la definición de una función propia termina en dólar el resultado debe ser ....

25. La definición de una función propia se hace mediante la instrucción ....

26. En la definición de las funciones propias de los nombres utilizados como .... no son variables del programa.

27. Las funciones .... se utilizan igual que las funciones intrínsecas.

28. Los argumentos, al utilizarse las funciones propias, deben .... en número, tipo y orden respecto a los utilizados en la definición.

29. La definición de una función propia puede utilizar .... del programa. Se distinguen porque no están en la lista de argumentos.

30. La .... de las instrucciones DEF en el programa depende de las versiones de BASIC.

Considere las siguientes definiciones de funciones:

```
10 DEF FNT(X) = 1./X
20 DEF FND(X,Y)= SQR(X*X+Y*Y)
30 DEF FNK$(X) = CHR$(X)+"HOLA"
40 DEF FNQ(X$,X)= X<LEN(X$)
50 DEF FNM(X$) = LEN(X$)-2
```

Encierre en un círculo la letra que corresponda a la alternativa correcta.

31. ¿Cuánto vale FND(3,4)?

a) 3.

b) 4.

c) 5.

d) 6.

32. Considere el programa siguiente:

```
100 LET R = 5
110 IF FNQ("HOLA",3) THEN LET R = R + 2
120 PRINT R
```

(Las instrucciones mencionadas más arriba se incluyen en este programa) ¿Qué número se imprime en la pantalla?

a) 5.

b) 7.

c) 2.

d) Error.

33. ¿Cuánto vale FNQ(3, «ADIOS»)?

a) 0.

b) 1.

c) −1.

d) Error.

34. ¿Cuánto vale 6.45*FNT(6.45)?

a) 0.

b) 1.

c) 2.

d) 3.

35. ¿Cuánto vale FNK$(33)?

a) HOLA.

b) ¡HOLA

c) HOLAHOLA.

d) " " (El texto vacío).

36. ¿Cuánto vale FNT(FNM («HOLA»))?

a) 0.5.

b) 0.25.

c) 0.75.

d) 1.

37. ¿Cuánto vale FND(1/SQR(2),1/SQR(2)) * 5?

a) 1.
b) 3.
c) 5.
d) 7.

38. ¿Cuánto vale FNM(«ADIOS»)/FNT(2)?

a) 20.
b) 30.
c) 6.
d) Error.

39. ¿Cuánto vale FNK$(«ADIOS»)?

a) HOLAADIOS
b) AHOLA
c) ADIOSHOLA
d) Es un error.

40. ¿Cuánto vale FNT(5,6)?

a) 0.2.
b) 0.3.
c) 0.4.
d) Es un error.



Capítulo 12
• Subrutina y números aleatorios

ESQUEMA DE CONTENIDO

Objetivos.	
Subrutinas.	Acceso a una subrutina. Retorno al programa principal. Modularización de los programas. Acceso de subrutina a subrutina. Bibliotecas de subrutinas.
Aplicaciones.	Diagrama de barras. Máximo común divisor. Listados.
Números aleatorios.	Generación de números aleatorios. Cebado inicial. Aplicaciones.

12.0 OBJETIVOS

El tema que se va a desarrollar en este capítulo es el más importante para la programación.

El objetivo final de la programación es resolver problemas complejos para ayudarnos a utilizarlos con el mínimo de esfuerzo.

Para que la resolución de un problema pueda realizarse libre de errores y lo más rápidamente posible es necesario descomponer el problema inicial en problemas más pequeños.

Este es el éxito de la ingeniería; atacar los problemas por partes y resolver los problemas resultantes cada vez más complicados. Piense, por ejemplo, en la construcción de un coche: Requiere construir muchísimas piezas y ensamblarlas adecuadamente.

En este aspecto, la programación no se diferencia en nada de la ingeniería. En el capítulo 7 ya hemos dado una primera introducción al tema.

Las instrucciones que se estudian en este capítulo son también instrucciones no estrictamente necesarias, pero que son indispensables para resolver programas de un grado de complejidad más elevado.

Se introduce el concepto de subrutina que, como indica la palabra, consiste en realizar una tarea subordinada a otras tareas.

Debe abordar este capítulo con la idea de que las instrucciones son sencillas pero que tienen un significado profundo, que cuesta de aprender y de manipular.

Hemos procurado ponerle en alerta sobre las líneas correctas de utilización de este tipo de instrucciones, pero una discusión amplia de las implicaciones que tienen en el diseño de un problema podría ser materia de toda una Enciclopedia.

Con esto queremos prevenirle de la sensación de que no hay dificultad ninguna; desconfíe de esta postura.

Tampoco caiga en el otro extremo, de ver dificultades en todas partes.

La mejor manera de aprender a utilizarlas es precisamente utilizándolas.

Cerramos el capítulo con la instrucción sobre números aleatorios.

La posibilidad de simular hechos reales en el ordenador, lleva acompañada la necesidad de utilizar lo que llamados suerte. Todas las cosas que suceden en la vida ordinaria tienen un conjunto de factores que no podemos controlar, que dan esta diversidad de hechos que agrupamos en la palabra suerte.

Es bastante corriente la opinión de que los números aleatorios sólo se utilizan en los programas de juegos. Ciertamente, en todos los programas de juegos suele haber un componente aleatorio, imprescindible para que funcionen.

Sin embargo, hay muchos programas muy serios, si es que los juegos no se consideran serios, en los que también intervienen.

La simulación del funcionamiento de un negocio, la simulación del funcionamiento de una nave industrial, de un vuelo, etc. son aplicaciones muy importantes de los números aleatorios.

Estudie esta función con atención, pues le será muy útil.

De todas maneras, la teoría de los números aleatorios es complicada.

Aquí sólo se ha explicado muy brevemente. No intente encontrar más que una aproximación al problema. Si desea profundizar, debe acudir a un texto de matemáticas o estadística que, como puede comprender, no es el objeto de esta Enciclopedia.



12.1 SUBRUTINA

Es frecuente que en los programas de cierta envergadura nos encontremos con instrucciones o, incluso grupos de instrucciones, que aparecen más de una vez a lo largo del listado. Sería interesante tomar estas instrucciones que se repiten y escribirlas una sola vez, formando un módulo que pudiera ser utilizado en aquellas partes del programa que lo requieran. De este modo, nos ahorraríamos el teclear varias instrucciones, evitando así posibles errores mecanográficos. A medida que aumenta el número de líneas de un programa, también aumenta la posibilidad de que cometamos errores al escribirlas. Por otra parte, obtendríamos un ahorro de memoria. Como ya sabemos, las instrucciones se almacenan en la memoria principal junto con los datos. Si evitamos la existencia de instrucciones repetidas, reduciremos el gasto de memoria.

El BASIC, al igual que la gran mayoría de lenguajes de programación, permite el empleo de un conjunto de instrucciones cuya ejecución puede desencadenarse desde cualquier punto del programa. Estos módulos o subprogramas reciben el nombre de *subrutinas*. Así pues, podemos decir que:

Definición de subrutina

> Una subrutina es un conjunto de instrucciones que realizan una tarea concreta.

Se denomina programa principal al conjunto de instrucciones que no forman parte de ninguna subrutina. De hecho, todos los programas que hemos visto hasta ahora eran programas principales.

El funcionamiento de un programa con subrutinas es el siguiente:

En aquel punto donde debe realizarse la tarea encomendada a la subrutina, se pasa control a la misma y simultáneamente se memoriza el lugar desde donde se produce la bifurcación. Cuando la subrutina ha terminado su ejecución, devuelve control a la posición del programa principal de donde procedía. Si a lo largo del programa se necesita realizar de nuevo la misma tarea, se le pasa control otra vez. Gracias a este sistema de memorización del punto de retorno, la subrutina devuelve siempre el control a la instrucción desde donde ha sido invocada. En la figura 1 se muestra un esquema del acceso a una subrutina. En la sección siguiente se describe con más detalle este proceso.

12.1.1 Acceso a una subrutina

Una subrutina en BASIC esta formada por el mismo tipo de instrucciones que el resto del programa. Acceder a una subrutina (o invocarla) significa pasar el control de la ejecución a la línea inicial de la misma.

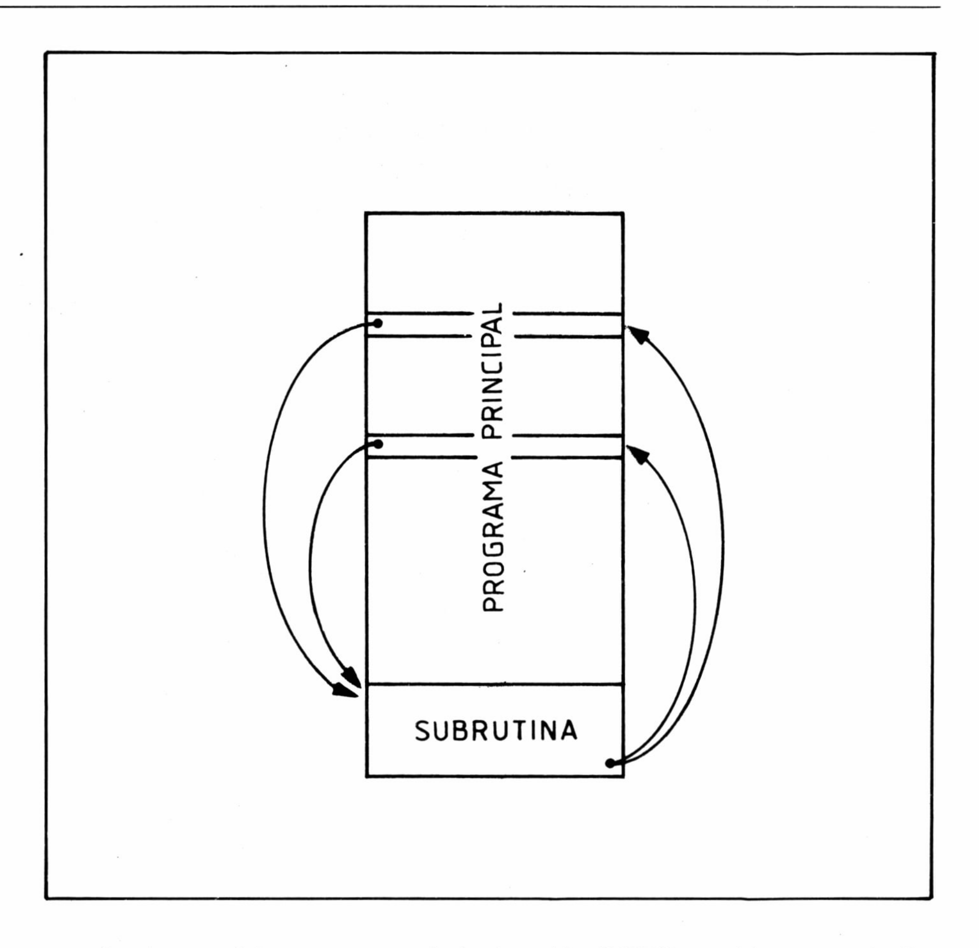


Figura 1 Esquema de un programa con una subrutina. Desde el programa principal se accede a la subrutina y desde ésta se retorna al punto de partida.

A primera vista parece que la instrucción GOTO serviría para este fin. Con la instrucción GOTO se pasa control al número de línea que deseemos.

Para acceder a una subrutina se usa el GOSUB

Pero para acceder a una subrutina necesitamos además otra cosa. Es necesario memorizar el número de línea desde donde la invocamos. Puesto que la instrucción GOTO no realiza esta función, debemos utilizar otra nueva. La instrucción que nos permitirá acceder a una subrutina es la instrucción GOSUB cuya sintaxis general es

GOSUB número de línea

La palabra GOSUB proviene del inglés «Go subroutine» que significa *ir a una subrutina*. Cuando se ejecuta la instrucción GOSUB, se pasa control al número de línea indicado. Este número debe corresponder al número de línea de la primera instrucción de la subrutina. Simultáneamente, la instrucción GOSUB memoriza su propio número de línea. Esta memorización es un proceso interno e invisible para el usuario.

12.1.2 Retorno al programa principal

Una vez finalizada la tarea, la subrutina debe retornar al programa principal. Necesitamos una instrucción que sepa consultar el número de línea

memorizado por la instrucción GOSUB y, de este modo, averiguar qué punto debe devolver el control. Para realizar esta función se utiliza la instrucción RETURN, cuya sintaxis general es muy simple ya que sólo consta de la palabra RETURN. La palabra inglesa RETURN significa retorno.

Para volver al programa principal se utiliza la instrucción RETURN

Cuando se ejecuta esta instrucción, se consulta el número de línea memorizado y se pasa control a la instrucción situada inmediatamente a continuación. Simultáneamente, se elimina el número de línea memorizado, puesto que ya ha sido utilizado y no es necesario conservarlo. Por esta razón, la instrucción RETURN sólo se puede emplear en el interior de una subrutina, ya que en un programa principal no encontraría memorizada la dirección de retorno, produciéndose un error. Podemos comprobarlo escribiendo en modo inmediato (sin número de línea) la instrucción RETURN. Al pulsar la tecla de fin de línea, el BASIC nos escribirá un mensaje de error indicando que no encuentra la dirección donde debe retornar.

En este punto conviene precisar un aspecto sobre esta instrucción. En la primera lección explicamos que la tecla de fin de línea podía venir indicada de varias maneras, como ENTER, como NEW-LINE o como RETURN, según el modelo de ordenador. No hay que confundir jamás el RETURN de la tecla de fin de línea con la instrucción de BASIC RETURN. Cuando nos referimos a la instrucción RETURN, esta palabra debe aparecer escrita con todas sus letras en la pantalla.

Veremos a continuación un ejemplo de cómo se utiliza una subrutina. Supongamos que deseamos escribir una tabla de raíces cuadradas con el siguiente formato:

NUM.	RAIZ
1	1
2	1.4142
.	.
.	.
.	.
10	3.1622

Como vemos, en la tabla aparecen tres líneas horizontales. Normalmente emplearíamos tres instrucciones PRINT iguales colocadas en los lugares oportunos del programa. Este es un ejemplo de una tarea repetida, aunque en este caso sea muy sencilla. Aprovecharemos ahora los conocimientos recién adquiridos y construiremos una subrutina que imprima una línea. El listado del programa será el siguiente:

```
10 GOSUB 200
20 PRINT "NUM.","RAIZ"
30 GOSUB 200
40 FOR I=1 TO 10
50   PRINT I,SQR(I)
60   NEXT I
70 GOSUB 200
80 END
```

```
200 REM Impresion linea
210 PRINT "--------------------"
220 RETURN
```

En la figura 2 presentamos el mismo programa dividido por bloques para indicar con claridad los lugares desde los cuales el programa principal remite a la subrutina.

La subrutina devuelve el control a la instrucción GOSUB de llamada mediante la instrucción RETURN.

Mecanismo de ida y vuelta

Las flechas de la figura 2 indican este ir del programa principal a la subrutina y de ésta al programa principal.

La tabla empieza con una línea horizontal, por tanto la primera instrucción será un GOSUB a la subrutina que imprime la primera raya de la tabla. Esta subrutina la hemos colocado en la línea 200. La primera instrucción de la subrutina es un REM con un comentario identificativo de la tarea que lleva a cabo. La colocación de estos comentarios es muy conveniente, sobre todo si el programa contiene muchas subrutinas, puesto que así se facilita la comprensión de su funcionamiento. En la línea 210 es donde se efectúa realmente la impresión de la raya.

Finalmente, en la línea 220, la instrucción RETURN devuelve control a la línea que sigue inmediatamente al GOSUB de la línea 10, es decir, que

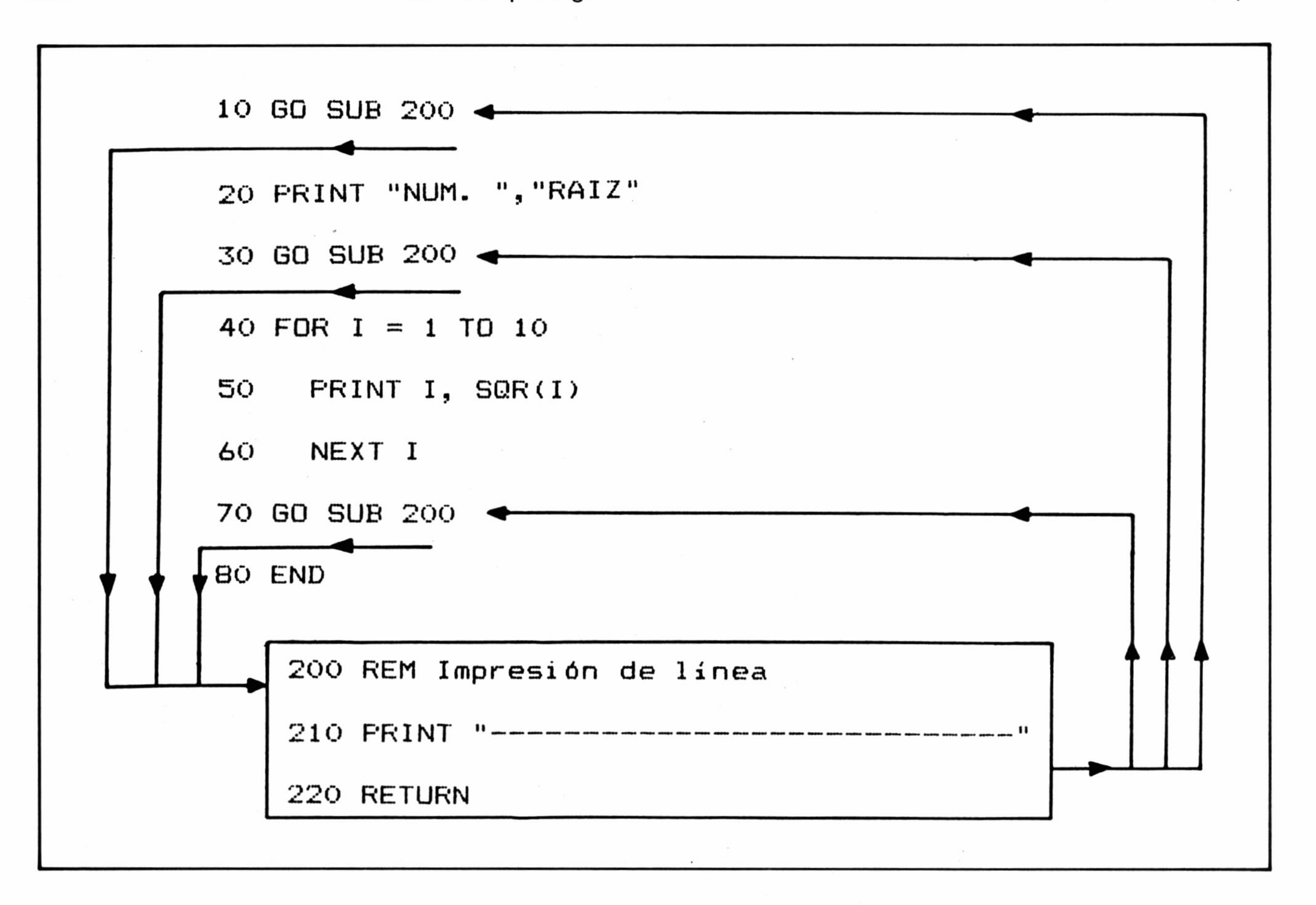


Figura 2 Esquema del funcionamiento de la subrutina en un programa real. Desde el programa principal se accede tres veces a la subrutina, y desde ésta se retorna al punto de partida.

el programa sigue por la línea 20 como indica la flecha. En ella se escribe el texto de la cabecera. A continuación necesitamos imprimir una nueva raya. Por consiguiente pasaremos control de nuevo a la subrutina 200. En este caso, la instrucción RETURN pasará control a la línea 40. Aquí empieza un proceso repetitivo que imprimirá los valores de la tabla. Para ello empleamos las instrucciones FOR/NEXT. En el interior de estas dos instrucciones se coloca la instrucción PRINT (línea 50) en donde se escribe el valor de la variable I y su raíz cuadrada. Al finalizar la tabla se necesita otra línea horizontal. Colocamos otra instrucción GOSUB para que la subrutina imprima una nueva línea.

Hay que separar las subrutinas del programa principal

Aunque este programa es muy sencillo, incluye varios aspectos que conviene resaltar. En primer lugar, las subrutinas se suelen colocar al final del listado, es decir, con un número de línea elevado. Además, aunque este número de línea puede ser cualquiera, se suele escoger un número redondo que sea fácil de recordar, como por ejemplo 200 o 1000. Ya hemos mencionado que es conveniente que la primera línea sea una instrucción REM que contenga un texto explicativo de la tarea que realiza la subrutina. En principio no hay limitación en el número de subrutinas que puede contener un programa. Únicamente existe el límite de la capacidad de memoria del ordenador. Para facilitar la comprensión del funcionamiento de un programa, se suele colocar al principio del listado (números de línea bajos) al programa principal y, a continuación, las subrutinas agrupadas.

De este modo se hace más evidente que su ejecución está subordinada al programa principal.

Otro punto importante a tener en cuenta es que para acceder a una subrutina hay que utilizar obligatoriamente una instrucción GOSUB. En caso contrario, se producirá un error cuando se ejecute el RETURN ya que no encontrará memorizada la línea de retorno. Por esta razón hay que separar mediante un END o un STOP el programa principal de las subrutinas. En el programa anterior, si no estuviese la instrucción END de la línea 80, la ejecución proseguiría por la línea siguiente (la 200), entrando en la subrutina de forma incorrecta (sin utilizar GOSUB) lo que daría lugar a un error. Puesto que generalmente, las subrutinas se colocan a continuación del programa principal, colocaremos siempre una instrucción END, STOP o GOTO al final del mismo.

12.1.3 Modularización de los programas

En el capítulo 7 del segundo tomo estudiamos las técnicas de diseño. En él vimos que para construir un programa era muy conveniente dividir el problema en varios bloques más sencillos. También estudiamos el diseño jerárquico que consistía en aumentar progresivamente el nivel de detalle de las operaciones que efectuaba cada bloque. Las subrutinas nos permitirán seguir con gran facilidad estas técnicas.

Subrutina es un elemento esencial para descomponer un programa en módulos

Una subrutina es un modulo que realiza una determinada tarea. Por tanto, podemos asimilarla a un bloque de proceso, del cual desconocemos los detalles por el momento. De esta forma descomponemos un programa en varios apartados, cada uno de los cuales se resolverá mediante una llamada a una subrutina. En el programa anterior hemos supuesto inicial-

mente que disponíamos de una subrutina que era capaz de imprimir una línea de guiones. Entonces, cada vez que en el programa principal hemos necesitado la impresión de una línea, nos hemos limitado a invocar la subrutina.

Ciertamente, en el programa anterior la subrutina realiza una tarea tan sencilla que no se aprecia ahorro de tiempo de escritura ni de líneas de programa. Pero incluso en este caso, el empleo de subrutinas nos reporta algunas ventajas.

En primer lugar, si construimos los diversos pasos de un programa a base de subrutinas, nos será más fácil resolverlo. Al concentrarnos en un grupo relativamente pequeño de instrucciones que componen una subrutina, tendremos menos dificultades en realizar un buen diseño. Entonces, una vez construida una subrutina, nos olvidamos de ella y pasamos a la siguiente, donde nos concentramos exclusivamente en la tarea que debe realizar ésta. Este proceso lo repetiremos hasta completar el programa.

Construir un programa de esta forma comporta dos ventajas adicionales:

— La primera de ellas es que los errores se localizan rápidamente. Puesto que cada subrutina realiza una tarea distinta, cuando se produzca un error en el funcionamiento del programa sabremos enseguida en qué subrutina se ha producido. Puesto que una subrutina contiene menos líneas que un programa completo, la detección y correción de errores será más cómoda. Por ejemplo, en el programa anterior, probablemente, la línea de guiones nos haya quedado demasiado corta o demasiado larga. Entonces sabemos enseguida que debemos intervenir en la subrutina 200 y, más concretamente, en la línea 210. Realizaremos la corrección oportuna, insertando o borrando guiones según el caso.

— La segunda ventaja, y muy importante, es que los módulos se pueden probar por separado, independientemente del programa principal y del resto de subrutinas. Supongamos que acabamos de corregir la línea 210 y queremos ver cómo funciona ahora. Para probarla utilizaremos una instrucción GOSUB inmediata, es decir, sin número de línea previo. Escribiremos:

```
GOSUB 200
```

De este modo accederemos directamente a la subrutina, la cual imprimirá la línea de guiones con la longitud corregida. Si la nueva línea no es todavía de nuestro agrado, la corregiremos otra vez y utilizaremos de nuevo la instrucción GOSUB para probarla. Fijémonos que no es posible utilizar la instrucción GOTO para probar una subrutina. Si escribimos

```
GOTO 200
```

se producirá un error en la instrucción RETURN.

Supongamos que ya hemos corregido la línea y que probamos todo el programa. Observamos que gracias al empleo de una subrutina, las tres líneas de que consta la tabla han quedado corregidas simultáneamente. Si en lugar de una subrutina hubiésemos utilizado tres instrucciones PRINT, nos hubiéramos visto obligados a repetir la corrección tres veces y a probar el programa globalmente. En este caso, el programa es muy sencillo y no cuesta demasiado probarlo completo.

La subrutina permite la prueba del programa y la detección de errores fácilmente

Sin embargo, en los programas muy largos puede resultar muy engorroso ejecutarlo por completo, únicamente para corregir un pequeño detalle. A esto nos referíamos cuando decíamos que trabajando con subrutinas es muy fácil localizar la causa de un funcionamiento incorrecto. Si en la tabla impresa las líneas no tienen la longitud adecuada, sabemos enseguida que hay que intervenir en la subrutina que la imprime. Además, al corregirla, todas las líneas de la tabla (que son tres) quedan automáticamente corregidas puesto que el mismo módulo es el que se encarga de dibujarlas todas.

Así por ejemplo, si en lugar de una línea de guiones, quisiéramos que aparecieran líneas formadas por asteriscos (*), únicamente tendríamos que cambiar la línea 210 por

```
210 PRINT "********************"
```

Una vez modificada la subrutina, veremos que al probar el programa, automáticamente las tres líneas estarán formadas por asteriscos (*).

De todo lo que hemos mencionado anteriormente, sacamos la conclusión de que una subrutina puede considerarse como una «caja negra». Esto significa que desde el punto de vista del programa principal, no nos importa cómo está construida la subrutina, ni qué instrucciones contiene. Simplemente nos interesa qué tarea realiza. Cuando necesitemos efectuar dicha tarea, realizaremos una llamada a la subrutina mediante la instrucción GOSUB.

Las subrutinas en BASIC utilizan las variables para comunicar los resultados

En la subrutina del ejemplo, su funcionamiento era autónomo y no necesitaba ninguna información procedente del programa principal. No siempre ocurre así. A menudo, la tarea que realiza la «caja negra» necesita conocer datos del programa principal. Asimismo, a veces la tarea consiste precisamente en efectuar algunos cálculos con datos procedentes del programa principal y devolver los resultados. Por tanto, habrá algunas variables que servirán para la comunicación de entrada-salida de la subrutina. De hecho, actuarán de modo parecido a los argumentos de las funciones. No debe extrañarnos el parecido entre funciones y subrutinas, pues ambas se pueden considerar como «cajas negras» que actúan donde se les invoca. No obstante hay algunas diferencias.

En primer lugar, todas las funciones devuelven un único valor como resultado, mientras que las subrutinas pueden devolver varios (o incluso ninguno como en el ejemplo que nos ocupa).

Además, los valores de los argumentos que se pasan a la función no pueden ser modificados por la propia función. Por ejemplo, en el segmento de programa siguiente

```
10 LET A=16
20 LET B=SQR(A)
30 PRINT A,B
```

está claro que la función SQR no modifica en absoluto el valor de A. Por el contrario, en las subrutinas está permitida la modificación de los valores de los argumentos de entrada. De hecho, una misma variable se puede utilizar para enviar datos a la subrutina y para que ésta los devuelva al programa principal.

A continuación veremos un par de ejemplos donde aplicaremos estos conceptos de variables de entrada-salida. El primero de ellos consistirá en una modificación del programa anterior, en el cual la subrutina 200 servirá para imprimir una línea formada por un símbolo cualquiera, guiones (−), signo igual (=), etc. El listado es el siguiente:

```
 10 LET A$="=": GOSUB 200
 20 PRINT "NUM.","RAIZ"
 30 LET A$="-": GOSUB 200
 40 FOR I=1 TO 10
 50   PRINT I,SQR(I)
 60   NEXT I
 70 LET A$="=": GOSUB 200
 80 END
200 REM Impresion linea
210 FOR J=1 TO 24
220   PRINT A$;
230   NEXT J
240 PRINT
250 RETURN
```

En este programa, cuando queremos trazar una línea, pasamos control a la subrutina 200 como antes. Pero ahora le pasamos la variable A$ que contiene el símbolo con el cual se deberá trazar la línea. Por tanto, A$ es el argumento de entrada a la subrutina. Como vemos, las rayas estarán formadas por un conjunto de signos igual (=), por guiones (−) y otra vez por signos igual (=), tal como se aprecia en la líneas 10, 30 y 70. Como es lógico, la subrutina estará construida de modo distinto. Para trazar la línea emplearemos un bucle, en el cual se imprimirá un símbolo cada vez. Colocamos además un punto y coma (;) al final de la instrucción PRINT para que los símbolos aparezcan impresos de forma contigua y no se produzca salto de línea. Las instrucciones 210 y 230 son las que establecen el lazo. La línea 220 es la que imprime el símbolo.

La subrutina es independiente del símbolo que contenga A$. Ella se limita a construir la línea con el símbolo contenido en A$ y que le será suministrado por el programa principal. Finalmente, la línea 240 se utiliza para que se produzca el salto de línea.

Si no estuviera presente, el cursor se quedaría justo al lado del último

símbolo de la línea y entonces el siguiente PRINT del programa principal empezaría en esta posición, originándose una tabla totalmente deformada. La línea 250 devuelve control al programa principal.

No olvidemos nunca que en BASIC, todas las variables son globales; es decir, que su ámbito de validez abarca al programa principal y a las subrutinas. Por tanto, sería incorrecto utilizar la variable I para el bucle de la línea 210, puesto que esta variable se usa ya en la línea 40.

Notemos que el programa principal no ha sufrido variación excepto por la variable A$, claro está. Puesto que lo único que se ha modificado es la impresión de las líneas de la tabla y de esta tarea se encargaba una subrutina, debemos introducir la modificación precisamente en esta subrutina, dejando prácticamente invariable el resto del programa.

El programa anterior contenía un ejemplo de subrutina con un argumento de entrada y ninguno de salida. En el siguiente ejemplo veremos un programa con una subrutina que tiene los dos tipos de argumentos.

Como sabemos, en BASIC los datos se encolumnan siempre por la izquierda. Si los datos son de tipo textual no hay ningún problema puesto que coincide con la forma normal de escribirlos. Por el contrario, los datos numéricos, si tienen distinto número de cifras, quedan mal encolumnados. Por ejemplo, el siguiente programa:

```
10 FOR I=0 TO 100 STEP 10
20   LET C=I*I
30   PRINT I,C
40   NEXT I
50 STOP
```

Escribirá una tabla con el siguiente formato:

```
0           0
10          100
20          400
30          900
40          1600
50          2500
60          3600
70          4900
80          6400
90          8100
100         10000
```

Como podemos apreciar, las dos columnas de números no quedan alineadas de la forma habitual. Algunas versiones sofisticadas del BASIC incluyen instrucciones especiales para solucionar esta cuestión. Pero si no disponemos de ellas, nos podemos construir una subrutina que nos resuelva el problema. El argumento de entrada de esta subrutina será el número que queremos encolumnar.

El argumento de salida será un texto que contendrá lás cifras del número, más la cantidad suficiente de espacios en blanco colocados a la izquierda. El listado del programa será el siguiente:

```
 10 FOR I=0 TO 100 STEP 10
 20   LET C=I*I: GOSUB 500
 30   PRINT I,A$
 40   NEXT I
 50 END
500 REM Alineado numeros
510 LET A$=STR$(C)
520 LET L=LEN(A$)
530 FOR J=L+1 TO 6
540   LET A$=" "+A$
550   NEXT J
560 RETURN
```

Como podemos apreciar, las dos columnas de números no quedan alineadas de la forma habitual. Algunas versiones sofisticadas del BASIC incluyen instrucciones especiales para solucionar esta cuestión. Pero si no disponemos de ellas, nos podemos construir una subrutina que nos resuelva el problema. El argumento de entrada de esta subrutina será el número que queremos encolumnar.

El argumento de salida será un texto que contendrá las cifras del número, más la cantidad suficiente de espacios en blanco colocados a la izquierda. El listado del programa será el siguiente:

En el programa principal sólo introducimos dos modificaciones. La primera consiste en pasar control a la subrutina 500 una vez calculado el valor del cuadrado *C*. La segunda modificación es sustituir el valor de *C* en la línea 30 por la variable que constituye el argumento de salida de la subrutina, en este caso *A$*. Analicemos un poco cómo funciona la subrutina 500.

La primera operación que se realiza es transformar el valor del número en un texto formado por sus cifras (línea 510). Para ello empleamos la función STR$. La variable *A$* contiene las cifras de *C* pero no sabemos cuántas. Para averiguarlo emplearemos otra función intrínseca, la función LEN. El resultado se almacena en la variable *L*. Para poder alinear los números necesitamos saber la anchura total de la columna que utilizamos para escribirlo. El número mayor que queremos escribir es 10000 que tiene cinco cifras. Para trabajar con comodidad, consideraremos una columna con un total de 6 posiciones. De acuerdo con esto, si un número tiene un total de *L* cifras, habrá que añadirle *6-L* espacios en blanco por la izquierda. Así, al número 100 con tres cifras le añadiremos tres espacios en blanco y al número 10000 sólo le añadiremos $6-5=1$ espacio.

Para hacer esta operación, utilizamos el bucle formado por las líneas 530 y 550. El bucle va desde a *L+1* que corresponde a la primera posición fuera de las cifras, hasta la longitud total de la columna que es 6. En la línea

540 se añaden los espacios en blanco a la variable *A$*. Mediante el símbolo de concatenación (+) añadimos espacios en blanco a la izquierda de *A$*. Una vez finalizado el bucle, *A$* contiene siempre un texto de longitud 6, con las cifras de *C* colocadas en su parte derecha.

Al probar el programa veremos que el resultado es el siguiente:

```
0            0
10         100
20         400
.            .
.            .
.            .
100      10000
```

Como es lógico, dado el planteo del programa, sólo ha quedado alineada la segunda columna. Ahora bien, gracias a la utilización de una subrutina para el alineado de números, si queremos alinear también la primera columna, la solución es extraordinariamente fácil. Tomemos la subrutina 500. Consideremos que se trata de una caja negra a la que se suministra *C* y nos devuelve *A$* con las cifras alineadas. Nada nos impide traspasar el valor de *I* a *C* y enviarlo a la subrutina la cual nos devolverá *A$* con las cifras alineadas. Almacenaremos este valor de *A$* en otra variable, por ejemplo *B$*, a fin de que no se destruya con el siguiente acceso a la subrutina.

El listado del programa principal quedaría de la siguiente manera:

```
10 FOR I=0 TO 100 STEP 10
20   LET C=I:GOSUB 500:LET B$=A$
30   LET C=I*I: GOSUB 500
40   PRINT B$,A$
50   NEXT I
60 END
```

Si ahora ejecutamos el programa, las dos columnas de números quedarán alineadas. Como se puede apreciar, la utilización de las subrutinas facilita enormemente la programación.

En este último programa, las dos columnas de números se construyen del mismo tamaño, 6 espacios. Para la primera columna este valor es algo grande. Se puede pensar en una modificación de modo que cada columna tenga su propia anchura. Además, a fin de aprovechar intensivamente la subrutina 500 escribiremos tres columnas, cada una con una anchura diferente. En la subrutina haremos únicamente una modificación. Cambiaremos el número 6 de la instrucción 530 por una *N*. La variable *N* contendrá la anchura de la columna correspondiente al actual de *C*. El listado completo será:

```
10 FOR I=0 TO 100 STEP 10
20   LET N=4 : LET C=I
```

```
 30    GOSUB 500 : LET B$=A$
 40    LET N=6 : LET C=I*I
 50    GOSUB 500 : LET C$=A$
 60    LET N=8 : LET C=I*I*I
 70    GOSUB 500
 80    PRINT B$;C$;A$
 90    NEXT I
100 END
500 REM Alineado numeros
510 LET A$=STR$(C)
520 LET L=LEN(A$)
530 FOR J=L+1 TO N
540   LET A$=" "+A$
550   NEXT J
560 RETURN
```

En este programa escribimos tres columnas que corresponden al valor de *I*, su cuadrado y el cubo. Las anchuras asignadas a cada columna son 4, 6 y 8 espacios respectivamente. Para cada columna hacemos las siguientes operaciones: Asignamos a *N* la anchura; en *C* colocamos el valor a encolumnar y pasamos control a la subrutina. Esta nos devuelve *A$*, que en los dos primeros casos necesitamos almacenarlo en variables auxiliares *B$* y *C$*. Finalmente, en la línea 80 escribimos los tres valores. Fijémosnos que utilizamos un punto y coma (;) para separar las tres variables. No es necesario utilizar la coma, puesto que las propias variables contienen el dato encolumnado. La subrutina 500 tiene dos argumentos de entrada, la variable *C* y la *N*. Tiene además, un argumento de salida formado por la variable *A$*.

Al ejecutar el programa veremos que se imprimen tres columnas de números perfectamente alineados por la derecha. En este punto conviene hacer una precisión. La subrutina 500 alineará los números de forma correcta únicamente si éstos son enteros, es decir, sin decimales. Si la variable *C* contiene un número fraccionario, la subrutina no tiene en cuenta el punto decimal y el número queda mal alineado. Mucho peor es el resultado si el número tiene el formato exponencial, como por ejemplo 12E+8. Construir una subrutina para alinear toda clase de números es algo complicado y, por otra parte, para la mayoría de casos, el programa anterior nos servirá perfectamente.

La dirección de retorno puede ser una instrucción intermedia en una sentencia

Un punto que habremos notado en los dos programas anteriores, es que la instrucción RETURN conoce la dirección de retorno no sólo respecto al número de línea, sino que también toma en consideración los dos puntos (:) que separan varias instrucciones dentro de una línea. Un ejemplo lo tenemos en la línea 20 del programa principal presentado.

```
20 LET C=I : GOSUB 500 : LET B$=A$
```

En ella, la instrucción GOSUB memoriza el número 20 (número de línea) y también que ella es la segunda instrucción de la línea. Entonces, la

instrucción RETURN devolverá control a la instrucción siguiente (no a la línea siguiente) efectuándose el LET de la variable B$. El comportamiento de la instrucción RETURN es distinto del de la instrucción GOTO, la cual se dirige siempre a un número de línea completo y la ejecución continúa por la primera instrucción de la línea.

El número de instrucciones RETURN en una subrutina no tiene límites

Una subrutina puede contener más de una instrucción RETURN. Siempre debe contener al menos una, pues de lo contrario no se producirá retorno al programa principal. Por ejemplo, en la subrutina de alinear números, puede ocurrir que el número tenga más cifras que la anchura de la columna. Esto se puede controlar; una posible actuación sería que en este caso añadiéramos un asterisco (*) al principio de las cifras para indicar este hecho. El nuevo listado de la subrutina sería el siguiente:

```
500 REM Alineado numeros
510  LET A$=STR$(C)
520  LET L=LEN(A$)
530  IF L>N THEN GOTO 580
540  FOR J=L+1 TO N
550      LET A$=" "+A$
560      NEXT J
570  RETURN
580  LET A$="*"+A$
590  RETURN
```

La instrucción IF de la línea 530 dirige el control a la línea 580 o a la 540 según L sea mayor que N o no. Si lo es, en la línea 580 se concatena un asterisco de aviso. En caso contrario se sigue el proceso ya conocido. Como vemos, la subrutina tiene dos puntos de retorno, las líneas 570 y 590. Por otra parte, la instrucción RETURN puede colocarse también en una línea múltiple. Utilizando esta posibilidad, la subrutina anterior queda más simplificada:

```
500 REM Alineado numeros
510  LET A$=STR$(C)
520  LET L=LEN(A$)
530  IF L>N THEN LET A$="*"+A$ : RETURN
540  FOR J=L+1 TO N
550      LET A$=" "+A$
560      NEXT J
570  RETURN
```

Las subrutinas se pueden probar de forma autónoma

En este caso, si la condición de la instrucción IF de la línea 530 se cumple, se pasa control a las instrucciones que siguen a la palabra THEN. Puesto que la última de ellas es un RETURN, las instrucciones siguientes (540 a 570) no tienen efecto.

Conviene insistir en que la utilización de subrutinas tiene la gran ventaja de que se pueden probar de forma autónoma. Además esta prueba puede realizarse en modo inmediato. Por ejemplo, si escribimos:

```
LET N=6 : LET C=25 : GOSUB 500 : PRINT A$
```

En pantalla aparecerá el número 25 precedido de cuatro espacios en blanco. Para comprobar que funcione en el caso de que el número de cifras supere a la anchura, escribiremos:

```
LET N=2 : LET C=459 : GOSUB 500 : PRINT A$
```

En este caso, en pantalla debe aparecer el resultado (el número 459) precedido de un asterisco. Una vez hemos comprobado que la subrutina funciona bien por separado, ya podemos utilizarla dentro de un programa principal.

El programa principal del caso anterior, lo modificaremos de modo que se empleen números más anchos que el tamaño de la columna.

En primer lugar modificaremos el bucle de la línea 10 a

```
10 FOR I=0 TO 500 STEP 30
```

Para leer con más claridad los números modificaremos también la línea 80 a

```
80 PRINT b$;" ";c$;" ";a$
```

de esta manera colocamos un espacio de separación entre ellos.

Ejecute el programa y observe que los últimos números (no podemos precisar cuántos porque depende de la versión del BASIC) de la tabla aparecerán con asteriscos porque su escritura requiere más espacios que la columna que hemos diseñado.

También observe que algunos números aparecen en formato exponencial o científico (tampoco podemos precisar cuántos, pues depende de la versión del BASIC).

El hecho de que escriba el número con formato exponencial no nos debe preocupar pues estamos pensando en alinear el texto del número por la derecha. El hecho que realmente interesa aquí es si cabe o no cabe en la zona que hemos reservado.

Observe también que cuando la subrutina nos pone un asterisco deja el resultado con una cadena de caracteres más larga que el ancho de la columna. Esto no es correcto ya que diseñamos la subrutina para asegurar que siempre nos devuelva el resultado con la longitud de la columna exacta.

Se puede evitar esto mediante la modificación de la instrucción 530 que debe quedar como

```
530 IF L>N THEN LET A$="*"+LEFT$(A$,N-1) :RETURN
```

de esta manera se aprecia la utilidad del asterisco, pues indica que el resultado no es correcto ya que se ha cortado para que se mantenga un resultado con el número de columnas deseado.

El *N-1* en la función LEFT$ debe colocarse para que la longitud sea de *N* exactamente ya que hemos añadido un asterisco.

12.1.4 Acceso de subrutina a subrutina

Desde una subrutina se puede pasar control a otra subrutina, actuando la primera como programa principal respecto a la segunda. A su vez, desde ésta se puede pasar control de nuevo a otra subrutina y así sucesivamente tantas veces como deseemos. El BASIC memoriza la nueva dirección de retorno, pero ¿qué ocurre con la memorización precedente? En realidad esta cuestión ya está prevista y la zona de memoria donde se almacenan las direcciones de retorno dispone de varias posiciones, habitualmente un total de 256. Entonces, cuando desde una subrutina se llama a otra, se memoriza la nueva posición de retorno sin destruir la anterior. Cuando se efectúa el RETURN de la última subrutina, se borra la última memorización. Las sucesivas instrucciones RETURN que se vayan efectuando, van eliminando las direcciones almacenadas hasta que al volver al programa principal esta zona de memoria quedará vacía.

El acceso a una subrutina se puede hacer desde otra subrutina

Veremos a continuación un ejemplo de acceso a una subrutina desde otra subrutina. En el programa anterior nos sería útil disponer de un módulo que nos separará grupos de tres cifras mediante puntos. Por ejemplo, el número veintisiete millones lo escribiría de la forma 27.000.000. Conviene remarcar que en BASIC este número sería erróneo, puesto que el punto se utiliza para separar los decimales. Para solucionar esta dificultad, almacenaremos las cifras junto con los puntos en una variable textual. Como ya sabemos, un dato textual puede contener cualquier conjunto de símbolos puesto que no son analizados.

Construiremos pues, una subrutina que realice esta función. El mecanismo básico consiste en rastrear el conjunto de cifras empezando por la derecha, e insertar un punto cada vez que la posición de las cifras sea múltiplo de tres. El listado de la subrutina es el siguiente:

```
600 REM Colocacion puntos
610  LET X$=""
620  FOR K=1 TO L
630      LET X$=MID$(A$,L-K+1,1)+X$
640      IF K=3*INT(K/3) AND K<>L THEN LET X$="."+X$
650      NEXT K
660  LET A$=X$ : LET L=LEN(A$)
670  RETURN
```

Al diseñar este módulo, hemos de tener en cuenta que se accederá a él desde la subrutina 500. Por tanto, los nombres de las variables que se utilizarán para la comunicación de datos deben coincidir. En este caso son dos, el conjunto de cifras contenido en *A$* y la lontitud almacenada en *L*. Otro punto a considerar es que no debe existir coincidencia de nombres entre las variables internas de este módulo y el resto de variables del programa.

El funcionamiento es el siguiente. En la línea 610 definimos una variable auxiliar, *X$*, donde iremos traspasando las cifras y los puntos de separación. Seguidamente estableceremos el bucle de rastreo. Utilizamos la variable K cuyo valor variará de 1 a *L*. En la línea 630 traspasamos una cifra de *A$* a la variable X$. La función MID$ nos extrae una cifra de *A$*. Puesto que el orden de extracción es de derecha a izquierda, la posición a extraer se calcula mediante la fórmula *L−K+1*, de modo que cuando *K* vale 1, la cifra es la que ocupa la posición *L* y cuando *K* vale *L*, la cifra es la que ocupa la posición 1. La línea 640 es la que decide si hay que colocar un punto. La primera parte de la pregunta K=3*INT(K/3) sirve para determinar si *K* es múltiplo de 3 como ya sabemos. En principio, si *K* es múltiplo de 3 hay que insertar un punto en *X$*. Sin embargo, hay que controlar que la inserción del último punto se realice únicamente si hay al menos una cifra más. De lo contrario, se producirían casos como el siguiente:

100000 se convertiría en .100.000

ya que el 1 ocupa la posición 6 que es múltiplo de 3. Para evitar este problema se añade una segunda parte a la pregunta de modo que no se produzca inserción si se trata de la última cifra. Una vez finalizado el bucle, debemos rehacer las variables de comunicación. Colocamos la variable *X$* en *A$* puesto que ésta es la que utiliza el módulo que lleva a esta subrutina. Asimismo, actualizamos el valor de *L*, puesto que la longitud de *A$* ha sufrido variación al añadir puntos entre las cifras.

Antes de retocar el programa principal, probaremos esta rutina por separado para comprobar su funcionamiento. Para ello, le pasaremos la variable *A$* con las cifras y *L* con el número de cifras.

Ejemplos:

```
LET A$="1000" : LET L=4 : GOSUB 600 : PRINT A$
```

El resultado será 1.000

```
LET A$="123456" : LET L=6 : GOSUB 600 : PRINT A$
```

El resultado será 123.456

```
LET A$="12345678" : LET L=8 : GOSUB 600 : PRINT A$
```

El resultado será 12.345.678

```
LET A$="543" : LET L=3 : GOSUB 600 : PRINT A$
```

El resultado será 543

Una vez realizadas las pruebas de comprobación, ha llegado el momento de modificar el programa inicial. La inserción de los puntos de separación deberá efectuarse después de convertir el número en su equivalente textual y antes de proceder a su encolumnado. Esto es lógico, puesto que la inserción de puntos modifica la longitud del número y por tanto disminuye la cantidad de espacios en blanco que se dejarán a su izquierda. De todo lo que acabamos de decir se deduce que este módulo deberá invocarse después de la línea 520 de la subrutina de alineado. Por tanto, en esta subrutina añadiremos la línea siguiente:

```
525   GOSUB 600
```

No hace falta añadir nada más, puesto que hemos tenido la precaución de escoger las variables de comunicación con los mismos nombres. Si ejecutamos ahora el programa completo veremos que se imprime una tabla en la cual los números están escritos en el formato al que estamos habituados. Antes habrá que colocar la línea 10 inicial, es decir:

```
10 FOR I=0 TO 100 STEP 10
```

La subrutina es el pilar del diseño modular

Acabamos de ver un ejemplo de cómo se llama a una subrutina desde otra. Pero además este programa constituye un buen ejemplo de cómo el diseño modular facilita enormemente la programación. La colocación de puntos para separar grupos de cifras es un problema que hemos resuelto por separado, construyendo un módulo que lo realizará. Esto nos ha permitido además comprobar de forma autónoma que el diseño era acertado y el funcionamiento correcto.

Una vez llegados a este punto nos podemos olvidar del funcionamiento interno de esta subrutina y considerarla como una «caja negra» a la cual se le pasa un conjunto de cifras en *A$*, y nos lo devuelve con las cifras separadas por puntos, sin importar cómo lo ha hecho. De esta forma, en la subrutina 500 utilizamos este módulo simplemente colocando la línea 525. Sabemos que después de efectuarse esta instrucción, la variable *A$* contendrá los puntos de separación y no hemos de preocuparnos en absoluto del modo cómo lo ha hecho.

Otro punto importante a considerar es que en el programa inicial sólo se ha efectuado una modificación (la línea 525) y en cambio, las tres columnas se han visto afectadas por la modificación. Esto se debe a que el proceso de alineado se efectúa siempre a través de las mismas subrutinas. Si en lugar de emplear una subrutina hubiéramos repetido las instrucciones para todas las columnas, el programa resultante sería tres veces más largo.

12.1.5 Bibliotecas de subrutinas

A menudo ocurre que en programas diferentes se emplean los mismos módulos, o al menos, parte de ellos. Este hecho ha originado la idea de construir una biblioteca de subrutinas. En esta biblioteca tendríamos una serie de subrutinas de uso general de modo que pudiéramos utilizarlas en varios programas. Las dos subrutinas del programa anterior constituyen un buen ejemplo de subrutinas de uso general. Las bibliotecas se almacenan en dispositivos magnéticos de modo que puedan cargarse en el ordenador, sin tener que teclearlas de nuevo. Una vez recuperadas del dispositivo de almacenamiento las subrutinas que necesitemos, se escribe el programa principal que las utiliza.

Puesto que en BASIC, la recuperación de instrucciones de un dispositivo magnético siempre es completa, conviene repartir las subrutinas en varias bibliotecas. En cada una de ellas colocaremos las subrutinas de características afines. Este procedimiento nos evitará que al recuperar una biblioteca nos encontremos con un gran número de subrutinas innecesarias en aquel programa concreto. A pesar de este reparto en varias bibliotecas, no siempre se puede evitar el hecho de que nos encontremos con subrutinas que no son necesarias en nuestro programa. En este caso las borraremos para que no consuman memoria ni alarguen innecesariamente el listado del programa.

Un buen conjunto de subrutinas repartidas en varias bibliotecas es de gran ayuda para programar con rapidez y fiabilidad. Es difícil dar normas generales sobre qué tipo de subrutinas se incluirán en bibliotecas. Esto depende de los intereses de cada usuario. No obstante se pueden dar algunas orientaciones generales.

Requisitos generales de las bibliotecas de subrutinas

1. Se mantendrá una lista con los números de línea por donde empieza cada subrutina. Se procurará que estos números sean exactos y espaciados de 100 en 100. Además serán números grandes (mayores de 1000 al menos), de modo que queden situados al final del programa.

2. Las subrutinas no deben contener demasiados números de línea, excepto en casos especiales. En todo caso, conviene subdividirla en submódulos.

3. Nunca se almacenará una subrutina en una biblioteca si no está comprobada exhaustivamente. En caso contrario, nos encontraríamos con que el error se ha propagado a todos los programas que la utilizan.

RESUMEN

La necesidad de ahorrar repeticiones de operaciones que se realizan frecuentemente nos obliga a considerar una idea de módulo.

Los módulos se denominan subrutinas, que son un conjunto de instrucciones que pueden ejecutarse desde cualquier punto del programa principal.

El conjunto de instrucciones que no pertenecen a ninguna subrutina se denomina programa principal.

El funcionamiento de las subrutinas es el siguiente:

— Desde cualquier punto de un programa se puede invocar a una subrutina.

— En este momento se accede al grupo de instrucciones que ha sido llamado y se ejecutan.

— Cuando termina este grupo de instrucciones se devuelve el control a la instrucción que la ha llamado.

Es necesario, para poder obtener un funcionamiento como el descrito, que se memorice la dirección de retorno, es decir, qué instrucción la ha llamado.

La invocación a una subrutina se hace mediante la instrucción:

GOSUB <número de línea>

El efecto de esta instrucción es semejante al GOTO pero con el efecto adicional invisible para el programador de memorizar desde qué punto del programa ha sido invocada.

Una vez transferido el control a las instrucciones que componen la subrutina se ejecutan normalmente.

La identificación para acabar una subrutina es la instrucción RETURN.

Esta instrucción tiene la misión de consultar el número de línea en la memoria de direcciones y realizar a su vez un GOTO a la línea memorizada, eliminando este número de línea de la memoria de direcciones.

Esta instrucción sólo puede utilizarse como final de una subrutina. Si se utiliza en el programa principal, no se encuentra línea de retorno memorizada y, por lo tanto, se comete un error.

La memorización de direcciones en la manipulación de subrutinas es más eficiente que en el caso del GOTO. Para el GOTO debemos enviar siempre el programa al inicio de una línea. En cambio, la instrucción RETURN devuelve el control mediante un GOTO a una instrucción que puede estar situada en el interior de una línea.

Este mecanismo está controlado por el lenguaje BASIC y es inaccesible para el programador.

En ningún caso se debe confundir la instrucción RETURN del BASIC con la tecla de fin de línea que en algunos ordenadores lleva el nombre de RETURN.

En BASIC es a veces difícil identificar dónde empiezan las subrutinas, pues, son números de líneas iguales que los del programa principal. Es conveniente, pues colocar los oportunos REM para identificar los inicios de las subrutinas y, a ser posible, agruparlas todas al final del programa.

Es mejor utilizar como numeración para la sentencia inicial de las subrutinas números fáciles de recordar tales como 100, 1000, 5500, etc.

Debe evitarse entrar en una subrutina si no es mediante la instrucción GOSUB pues al alcanzarse la instrucción RETURN se produce un error.

La separación del programa principal del bloque de subrutinas se hace mediante una instrucción END o STOP. Si no se coloca, el programa continúa por las líneas de la primera subrutina hasta alcanzar inevitablemente el RETURN y obtener un error, además de haber ejecutado unas sentencias innecesarias.

La subrutina es la herramienta básica para la descomposición del programa en procesos.

Permite que nos concentremos en pocas instrucciones.

Cuando tenemos errores es más fácil de detectar dónde están y corregirlos.

Cuando se corrige un error en una subrutina se corrige en todos los lugares que la utilizamos. Por eso, aunque las tareas sean muy sencillas, la utilización de subrutinas reporta ventajas.

La subrutina se comporta, desde el punto de vista del programa principal, como el realizador de tareas completas, o caja negra, por lo que no importa cómo está construida ni qué instrucciones contiene, nos interesa solamente la tarea que realiza.

La comunicación de datos entre el programa principal y las subrutinas se hace mediante argumentos que, a diferencia de las funciones, son variables del programa que el programador destina a tal efecto.

Las variables que se utilizan como argumento pueden modificarse en la subrutina; esto las diferencia de las funciones.

Esta manera de pasar argumentos, de hecho utiliza las mismas variables que el BASIC, es cómoda pero tiene sus riesgos sobre todo en la utilización de las variables de control de los bucles.

Una subrutina puede contener más de una instrucción RETURN, aunque al ejecutarse sólo actúa uno.

La utilización de subrutinas no está limitada a un nivel sino que puede llamarse a una subrutina desde otra subrutina.

Esta posibilidad ya está prevista en la memorización de direcciones. Se almacenan de tal manera que el RETURN toma la última dirección memorizada y la elimina con lo que la nueva dirección de retorno es la anterior de la que se ha utilizado más recientemente.

En cierto modo, las subrutinas permiten que el programador aumente las instrucciones elementales que tiene la máquina. Construye tareas más complicadas que puede utilizar en diversos programas.

El almacenamiento en registros magnéticos de estas subrutinas se denomina biblioteca de subrutinas.

Aunque no se almacenen en registro magnético también son útiles, pues permiten eliminar el análisis de un problema que ya hemos resuelto otras veces con el consiguiente ahorro de tiempo y permiten un desarrollo más rápido de un programa.

El requisito más importante que deben tener las bibliotecas de subrutinas es que cada una de ellas se pruebe exhaustivamente; es decir, que se pueda asegurar que están libres de error.

En caso contrario, el supuesto ahorro de tiempo puede convertirse en un dispendio debido a que creemos que la subrutina es correcta.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

1. Se denomina .... a un conjunto de instrucciones que pueden ejecutarse varias veces y se llaman desde diversos puntos del programa principal.

2. Para poder realizar este mecanismo de ida y vuelta es necesario que se .... la instrucción que realiza la llamada.

3. La instrucción GO SUB es semejante al GO TO excepto que la primera realiza la memorización de la .... que la ha llamado.

4. La instrucción .... es la indicación de que la subrutina ha finalizado y debe transferir el control a la instrucción que ha llamado a la subrutina.

5. La utilización de la instrucción RETURN en un programa principal es un .... ya que no existe instrucción memorizada para realizar el retorno.

6. El mecanismo de memorización de la instrucción de llamada es .... al programador y lo realiza el lenguaje BASIC de manera automática.

7. Dentro de una subrutina pueden existir .... de una instrucción RETURN.

8. La subrutina es el método básico de .... un programa en procesos distintos.

9. Los .... de una subrutina son las mismas variables del BASIC.

10. Las subrutinas cumplen .... completas y desde el programa principal se tratan como cajas negras.

De las afirmaciones siguientes rodee con un círculo la S si es aplicable a la instrucción GO SUB y con una T si es aplicable a la instrucción GO TO.

Tenga en cuenta que hay afirmaciones aplicables a ambas, con lo que debe rodear con un círculo la S y la T y hay afirmaciones que no son aplicables a ninguna de las dos instrucciones con lo que no debe señalar nada.

11. Transfiere el control a la instrucción señalada. S T

12. Vuelve a la instrucción de llamada cuando encuentra una instrucción RETURN. S T

13. Sólo transfiere el control al inicio de una línea de programa. S T

14. El retorno se hace en una instrucción aunque esté en la mitad de una línea. S T

15. Sólo se puede utilizar en el programa principal. S T

Encierre con un círculo la letra que corresponda a la respuesta correcta.

16. La comunicación de los *datos* entre el programa principal y las subrutinas se hace mediante:

    a) Las variables del programa.
    b) Argumentos como en las funciones.
    c) Mediante variables que empiecen por a.
    d) Mediante un número de instrucción.

17. Para evitar errores es necesario separar el programa principal de las subrutinas mediante la instrucción.

    a) PRINT.
    b) NEW.
    c) END.
    d) LIST.

18. Para distinguir bien lo que son subrutinas de las instrucciones que componen el programa principal es conveniente iniciar las subrutinas con la instrucción.

    a) STOP.

    b) DIM.

    c) CLS.

    d) REM.

19. ¿Cuál de las llamadas siguientes a una subrutina es correcta?

    a) GO SUB 1000+23.5

    b) GO SUB 1000 : LET i = 23.5

    c) LET ZETA = 100 : LET EQUIS = 900: GO SUB ZETA+EQUIS

    d) GO SUB MIL$

20. El requisito más importante de una biblioteca de subrutinas es que:

    a) Tenga muchas.

    b) Sean cortas.

    c) Tengan muchos comentarios.

    d) Estén libres de error.

12.2 APLICACIONES

Veremos a continuación una serie de programas que utilizan subrutinas.

12.2.1 Diagrama de barras

Frecuentemente nos gustaría presentar los resultados obtenidos por un programa en forma gráfica, en lugar de utilizar la consabida tabla de números. Aunque la mayoría de ordenadores incorporan algún sistema de gráficos de alta resolución, éstos no son necesarios para construir dibujos de tipo sencillo.

El programa construirá una barra formada por asteriscos (*) cuya longitud será proporcional al valor del dato. Por tanto, la primera tarea a rea-

lizar será la elaboración de una subrutina para construir la barra. El listado de esta subrutina es el siguiente:

```
500 REM Construccion barra
510 LET B$=""
520 FOR B=1 TO L
530  LET B$=B$+"*"
540  NEXT B
550 RETURN
```

Como de costumbre, en la primera línea colocamos un comentario informativo. La variable *B$* será la que contendrá la barra. Por consiguiente, la primera operación consistirá en asegurar que está vacía. A continuación se empieza un bucle de 1 a *L*, en donde *L* contiene el valor de la longitud de la barra. En la línea 530 se va concatenando un asterisco (*) a *B$*. La línea 540 es la que cierra el bucle y seguidamente viene la instrucción RETURN.

Pruebas de subrutinas

Antes de seguir, hemos de asegurar que la subrutina funciona perfectamente. Como siempre, para verificarla efectuaremos pruebas en modo inmediato. Las variables de comunicación son dos. Desde el programa principal se le pasa el valor de *L*, es decir, la longitud de la barra. A su vez, la subrutina devuelve la variable *B$* que contiene precisamente *L* asteriscos (*). Escribamos

```
LET L=10: GOSUB 500: PRINT B$
```

En pantalla debe aparecer una línea formada por diez asteriscos. Es conveniente efectuar varias pruebas para distintos valores de L. Una vez verificada la subrutina pasaremos a construir el programa principal. Este programa tendrá dos partes. En la primera de ellas se realizará la lectura de los datos. En la segunda, se dibujarán las barras. El listado, que se añadirá a la subrutina, es el siguiente:

```
10 REM Graficos barras
20 LET N=3: DIM A(N)
30 FOR I=1 TO N
40    INPUT A(I)
50    NEXT I
60 INPUT "MAXIMO:",M
70 FOR I=1 TO N
80    LET L=A(I)*20/M
90    GOSUB 500
100   PRINT I;TAB(5);B$
110   NEXT I
120 END
```

En la línea 20 se define un conjunto dimensionado *A* de 3 elementos. Las líneas 30, 40 y 50 constituyen el conocido lazo para llenar el conjunto. En la línea 60, el programa nos pide el valor máximo para el que se construirá el gráfico. Este valor se almacena en la variable *M*. A continuación viene el bucle para dibujar las barras. En la línea 80 se efectúa una conversión de escala a fin de adaptar los valores al tamaño de la pantalla. La longitud de la barra variará de cero a 20 asteriscos (*). Por el contrario, los números almacenados *A* pueden ser cualesquiera, por ejemplo 1000, 1500 y 1200. Es obvio que necesitamos transformar estos números a la escala de 1 a 20. Para esto sirve la variable *M*. Si para aquellos valores de *A* hubiésemos entrado M=2000, entonces los correspondientes valores de *L* serían 10, 15 y 12. A continuación pasamos control a la subrutina 500 donde se construye la barra. El retorno se produce en la línea 100 donde se imprime el valor de *I* (número de orden) y la variable *B$* que contiene la línea de asteriscos (*). Finalmente la línea 110 cierra el lazo.

Si al ejecutar el programa entramos los valores comentados anteriormente, se imprimirán en pantalla tres barras de longitudes de 10, 15 y 12. Si repetimos el proceso empleando un valor de *M* mayor (por ejemplo 5000) entonces las barras serán de inferior longitud.

Para representar simultáneamente en número mayor de datos, se cambiará el valor de *N* de la línea 20. Este programa tiene el inconveniente de que debemos suministrarle nosotros el valor máximo de la escala. En el capítulo dedicado a conjuntos dimensionados estudiamos el procedimiento para que el propio programa averigue el máximo de una lista. Después de introducir esta variante, el nuevo listado será el siguiente:

```
10 REM Grafico barras
20 LET N=3: DIM A(N)
30 FOR I=1 TO N
40   INPUT A(I)
50   NEXT I
55 LET M=A(1)
60 FOR I=2 TO N
70   IF A(I)>M THEN LET M=A(I)
80   NEXT I
90 FOR I=1 TO N
100  LET L=A(I)*20/M
110  GOSUB 500
120  PRINT I;TAB(5);B$
130  NEXT I
140 END
```

Las líneas añadidas o modificadas son las que están comprendidas entre la línea 50 y la 70. El resto del programa no sufre variación. Estas líneas son las que sustituyen la entrada manual de la variable *M* por el cálculo automático. Cuando el programa llegue a la línea 70, en *M* habrá el valor máximo del conjunto *A*. A causa de este sistema de calcular *M*, la

barra que corresponda al elemento máximo de *A* será de longitud 20. En el caso anterior, la longitud dependía del valor de *M*. Gracias a este sistema de cálculo podemos introducir cualquier conjunto de datos puesto que el programa realizará el escalado de forma automática.

12.2.2 Máximo común divisor

Este programa es de tipo matemático. Se trata de hallar el máximo común divisor de dos números. Ya sabe que el máximo común divisor de dos números es el mayor número que es capaz de dividir los números iniciales exactamente. Siempre es posible encontrar este número ya que por lo menos el 1 divide a cualquiera de los dos números iniciales.

Para ello emplearemos el conocido algoritmo de Euclides desarrollado en el siglo III a.C. Este método se basa en realizar sucesivas divisiones enteras de los dos números.

En primer lugar construiremos una subrutina que calcule el máximo común divisor de X e Y. El listado es el siguiente:

```
300 REM Algoritmo de Euclides
310 LET Q=INT(X/Y)
320 LET R=X-Q*Y
330 IF R=0 THEN RETURN
340 LET X=Y: LET Y=R
350 GOTO 310
```

En la primera línea de la subrutina, en este caso la 300, se coloca el título de la misma. A continuación, efectuamos la división entera de *X* por *Y*. Para ello empleamos la función INT que nos suprime los decimales. El resultado es el cociente de la división y se almacena en *Q*. A continuación, calculamos el resto *R* de la división. Si este resto es cero, el proceso ha terminado, puesto que implica que la división ha sido exacta y por tanto *Y* es un divisor de *X*. Si no es así, efectuamos entonces la operación clave del algoritmo: En *X* colocamos el valor de *Y* y, a la vez, en *Y* colocamos el valor del resto. Seguidamente pasamos control de nuevo a la línea donde se efectúa la división (310). Este proceso se repite tantas veces como sea necesario hasta que el resto *R* sea igual a cero. Observamos que este lazo siempre tiene salida ya que cuando *Y* valga 1, el resto será forzosamente cero. Cuando la subrutina finaliza, en Y se encuentra el máximo común divisor.

Las variables de entrada son *X* e *Y*. Los valores originales de *X* e *Y* son destruidos durante la ejecución. Cuando se produce el retorno al programa principal, la variable *Y* contiene el resultado y por tanto actúa de variable de salida. El listado del programa principal que se añadirá a la subrutina, es el siguiente:

```
10 REM Maximo Comun Divisor
20 INPUT A
30 INPUT B
```

```
40 LET X=A: LET Y=B
50 GOSUB 300
60 PRINT "EL M.C.D. DE ";A;" Y ";B;" ES ";Y
70 STOP
```

En las líneas 20 y 30 se produce la entrada de datos. A continuación, en la línea 40 se traspasan los valores entrados a las variables *X* e *Y* que actúan de argumentos de entrada para la subrutina, a la cual accedemos mediante la línea 50. Cuando se produce el retorno escribimos el resultado contenido en *Y*. Finalmente en la línea 70 se detiene el proceso.

Probablemente nos preguntamos por qué se leen las variables *A* y *B* y luego traspasamos sus valores a *X* e *Y*, en lugar de leer directamente estas dos últimas variables. La razón es simple. Recordemos que la subrutina destruye los valores originales de los argumentos. Entonces en la línea 60 únicamente podríamos escribir el valor de Y pero no los datos de partida. Es por esta razón que mantenemos una copia de los datos originales en *A* y *B*.

Ejecutemos el programa y realicemos las siguientes pruebas:

Para A=35 y B=290 en pantalla aparecerá

El M.C.D. DE 35 Y 290 ES 5

Para A=32 y B=384 el ordenador escribirá

EL M.C.D. de 32 y 384 es 32

Para A=19228 y B=32436 el mensaje en pantalla será

EL M.C.D. DE 19228 Y 32436 ES 4

Podríamos preguntarnos si en este caso no hubiera sido más práctico construir el programa de una sola pieza sin utilizar subrutinas. Ciertamente, si sólo hay que calcular el máximo común divisor de dos números, la utilización de la subrutina es innecesaria. Por el contrario, si queremos hallar el máximo común divisor de tres o más números, el disponer de un módulo que efectue el algoritmo de Euclides es muy útil. Veamos cuál es el listado del programa principal para calcular el máximo común divisor de tres números.

```
10 REM Maximo Comun Divisor
20 INPUT A
30 INPUT B
40 INPUT C
50 LET X=A: LET Y=B
60 GOSUB 300
70 LET X=C
80 GOSUB 300
```

```
 90 PRINT A;TAB(10);B;TAB(20);C
100 PRINT "M.C.D.=";Y
110 STOP
```

En este caso, la entrada de datos formada por las líneas 20, 30 y 40. Los tres datos de partida se almacenan en las variables A, B y C. En primer lugar hay que calcular el máximo común divisor de los dos primeros datos. Por tanto, traspasamos los valores de *A* y *B* a las variables *X* e *Y* respectivamente. A continuación pasamos control a la subrutina 300. Esta nos devuelve en *Y* el máximo común divisor de *A* y *B*. El resultado final será precisamente el máximo común divisor de *Y* y el tercer dato. Por tanto, en *X* colocamos el valor de *C* (línea 70) e invocamos de nuevo la subrutina 300. De nuevo, ésta nos devuelve el resultado almacenado en *Y*. Seguidamente, se escriben los valores iniciales y el resultado mediante las líneas 90 y 100. Finalmente, en la línea 100 se detiene el proceso.

Efectuemos ahora algunas pruebas. Escribamos el comando RUN y entremos algunos valores de A, B y C.

Para A=60, B=90 y C=120, en pantalla aparecerá

MCD=30

Para A=32, B=384 y C=72, el resultado que escribirá el programa es

MCD=8

Como hemos podido observar, el empleo de una subrutina nos ha permitido ampliar el programa escribiendo únicamente unas pocas instrucciones.

Hay que evitar la descomposición modular excesiva

Hasta ahora hemos visto las grandes ventajas que se derivan de la utilización de subrutina. Sin embargo, cuando se diseña un programa hay que tener la precaución de construir módulos que realicen operaciones útiles. Hay que evitar caer en la tentación de efectuar una excesiva modularización del programa puesto que entonces disminuye la velocidad de ejecución. No olvidemos que cada vez que se acude a una subrutina se efectúan las siguientes operaciones:

— Memorización de la dirección de retorno.

— Salto a la primera línea de la subrutina.

— Retorno al programa principal y

— Borrado de la dirección memorizada.

Si el programa está bien diseñado, este pequeño gasto extra de tiempo de ejecución queda sobradamente compensado por las ventajas que comporta el empleo de subrutinas. Es difícil dar normas para construir módulos útiles, pues dependen de cada caso concreto. En todo caso será útil repasar las técnicas de diseño estudiadas en el capítulo 7.

Por otra parte, la experiencia que se va adquiriendo al realizar programas será nuestra mejor aliada para diseñar correctamente los programas.

A continuación veremos un pequeño programa que pone de manifiesto la pérdida de tiempo que se produce a causa de una excesiva modularización.

```
10 FOR I=1 TO 1000
20 LET A=3+4
30 NEXT I
40 PRINT "FIN"
```

En este programa se realiza un bucle en el que se repite 1000 veces la misma operación, que en este caso es una simple suma. Este programa sirve para evaluar la velocidad a la cual suma nuestro ordenador. Tomaremos un reloj o un cronómetro y mediremos el tiempo transcurrido desde la orden de RUN hasta que la palabra FIN aparece en pantalla. Según el modelo de ordenador, este proceso puede se muy rápido y entonces no mediríamos un tiempo significativo. En este caso cambiaremos el 1000 por un número mayor. Lo ideal sería que transcurrieran al menos 10 segundos. Es conveniente realizar varias ejecuciones y realizar la media de los tiempos tomados. A continuación construiremos el programa con una subrutina. El listado es el siguiente:

```
 10 FOR I=1 TO 1000
 20 GOSUB 100
 30 NEXT I
 40 PRINT "FIN"
 50 STOP
100 LET A=3+4
110 RETURN
```

Como vemos, el programa es el mismo. Unicamente ha cambiado el hecho de que la suma se realiza en un módulo aparte. Si ejecutamos este programa, obtendremos unos tiempos significativamente mayores que en el caso anterior. Este es un ejemplo claro de cómo una excesiva modularización ocasiona pérdidas de tiempo sin reportar a cambio beneficio alguno.

12.2.3 Listados

La confección de listados, ya sea por pantalla o por impresora es una tarea de las más frecuentes en informática. Veremos a continuación, cuáles

son las técnicas que deben seguirse para confeccionar listados con formato completo. En un listado se suelen distinguir tres partes:

— La primera de ellas es la cabecera que identifica la naturaleza del listado.

— La segunda es el cuerpo del listado y

— La tercera es el pie de página.

Tanto la cabecera como el pie de página se imprimen una sola vez, mientras que el cuerpo del listado está formado por una repetición de líneas iguales o parecidas.

Como ejemplo de listado en formato completo, construiremos una tabla de raíces cuadradas. La cabecera y el pie de página las construiremos con subrutinas separadas. Antes de construirlas debemos establecer tres variables importantes. La variable *P* contendrá el número de página que se está imprimiendo. La variable *LM* contiene el número máximo de líneas por página y la variable *L* almacena el número de línea actual dentro de la página.

Cabecera de subrutina

Según lo anterior, el listado de la subrutina que imprimirá la cabecera será el siguiente:

```
300 REM Cabecera
310 LET P=P+1: CLS
320 PRINT "TABLA RAICES     PAGINA: ";P
330 PRINT "---------------------------"
340 PRINT " NUMERO","RAIZ"
350 PRINT "---------------------------"
360 LET L=4
370 RETURN
```

Aparte de las instrucciones PRINT que imprimen los textos fijos, existen dos instrucciones importantes. La primera es la línea 310 en donde se incrementa el valor de *P*, es decir el número de página. Este valor se emplea en la línea 320. Además, en esta línea se borra pantalla. La otra instrucción importante se encuentra en la línea 360. En ella establecemos que la cabecera ya ha gastado cuatro líneas de la página.

Subrutina pie

El listado de la subrutina que imprimirá el pie de página es el siguiente:

```
400 REM Pie de pagina
410 PRINT "---------------------------"
420 LET A$=INKEY$
430 IF INKEY$="" THEN GOTO 420
440 RETURN
```

Esta subrutina tiene dos misiones. La primera de ellas es escribir una línea que cierre la página. La segunda es hacer que el ordenador se espere hasta que el operador haya tenido tiempo de leer la página y dé la orden de seguir. Esto se consigue mediante la función INKEY$, que bloquea la ejecución entre las líneas 430 y 440 hasta que el operador pulsa una tecla.

El programa principal tiene el siguiente listado que añadiremos a las subrutinas anteriores:

```
 10 REM Listado
 20 LET P=0: LET LM=15
 30 GOSUB 300
 40 FOR I=1 TO 100
 50   PRINT I,SQR(I)
 60   LET L=L+1
 70   IF L>LM THEN GOSUB 400: GOSUB 300
 80   NEXT I
 90 GOSUB 400
100 STOP
```

En la línea 20 establecemos los datos iniciales. El número de página vale cero y el número de líneas por página lo hacemos igual a 15. La siguiente operación es imprimir la cabecera y por tanto pasamos control a la subrutina 300. A continuación viene el cuerpo del listado que se construye con un bucle FOR/NEXT formado por las líneas 40 y 80. En el interior del lazo, además de la impresión de las raíces (línea 50), realizamos otra operación consistente en el control del número de líneas escritas. A tal fin, incrementamos el valor de *L* cada vez que se imprime una línea. A la vez, preguntamos si este número de línea supera el máximo admitido por página. Si es así, pasamos control a la subrutina del pie de página, la cual escribirá una línea de cierre y se esperará a que pulsemos una tecla para seguir. A continuación se pasa control a la subrutina que borra la pantalla e imprime una nueva cabecera. Además incrementa el número de página *P* y coloca un 4 en la variable *L*. Este proceso se va repitiendo hasta que finaliza el listado. En ese momento pasamos control por última vez a la subrutina 400 que escribe el pie de página.

Si queremos utilizar este programa para que el listado salga por impresora, hay que introducir algunas modificaciones. El primer paso es cambiar todas las instrucciones PRINT por LPRINT. Además, suprimiremos las instrucciones 420 y 430 ya que ahora carecen de sentido. Puesto que el listado aparece impreso en papel, no tiene objeto esperar a que el operador haya leído una página para imprimir la siguiente. Un último punto a considerar es la instrucción CLS de la línea 310. Esta instrucción la utilizamos para empezar página nueva. En la impresora esto significa que debe producirse un salto de página. Podemos encontrarnos con dos situaciones según el modelo de impresora. Hay algunas que realizan salto de página y otras que sólo admiten la impresión continua. En el primer caso, cambiaremos la instrucción CLS por

Salto de página en las impresoras

```
LPRINT CHR$(12)
```

ya que el 12 es el código ASCII que produce el salto de página.

Si sólo admite la impresión continua, entonces en la subrutina 400 colocaremos una serie de instrucciones LPRINT vacías para producir unos cuantos saltos de línea de modo que quede una separación suficiente entre el pie de una página y la cabecera de la siguiente. Frecuentemente, en los listados se suele llevar un total acumulado de alguna columna. Este total se imprime al final del listado. En este caso hay que construir dos subrutinas de pie de página distintas. Una de ellas servirá para los saltos de página durante el listado y la otra servirá para el último pie de página. El listado del programa principal será el siguiente:

```
 10 REM Listado
 20 LET P=0: LET LM=15
 30 LET T=0
 40 GOSUB 300
 50 FOR I=1 TO 100
 60    PRINT I,SQR(I)
 70    LET T=T+SQR(I)
 80    LET L=L+1
 90    IF L>LM THEN GOSUB 400: GOSUB 300
100    NEXT I
110 GOSUB 500
120 STOP
```

Definimos la variable *T* que servirá para llevar el total acumulado de la columna de raíces. Antes de empezar el listado, se inicializa con el valor cero. Por cada línea que escribe, se acumula el valor de la raíz en *T*, tal como se observa en la línea 70. Cuando finaliza el listado, pasamos control a la subrutina 500 (línea 110) en lugar de a la 400. El resto del listado es idéntico al programa anterior, así como las dos subrutinas 300 y 400. En cuanto a la nueva subrutina 500, el listado es:

```
500 REM Total
510 PRINT "-----------------------------"
520 PRINT " TOTAL...";T
530 RETURN
```

Al ejecutar este programa, en la última página se escribirá el total de la segunda columna.

12.3 NUMEROS ALEATORIOS

Un uso importante de los ordenadores es la imitación de un proceso de la vida real. Esto se conoce como simulación. Puesto que tales procesos

dependen casi siempre del resultado de sucesos aleatorios, necesitamos algún procedimiento para obtener números al azar. En realidad, la obtención de números aleatorios es algo que va en contra de la propia naturaleza de los ordenadores. Para que se dé la existencia del azar, es necesario que haya algunas variables del proceso que escapen a nuestro control. Un ordenador es una máquina totalmente determinista, es decir, que a igualdad de causas produce iguales efectos y, por tanto, no puede existir ninguna variable que no se pueda controlar. Como conclusión, no se puede obtener un azar auténtico usando un ordenador. No obstante, esto no significa que debamos renunciar a obtener números con un azar más o menos imperfecto. Para ello se usa un procedimiento muy ingenioso que describiremos a continuación.

Modo básico de cálculo

En primer lugar hay que advertir que la obtención de números aleatorios es un proceso interno del ordenador y la explicación que daremos seguidamente sólo pretende dar el fundamento en el que se basa este cálculo. El mecanismo típico consiste en hallar restos de divisiones. Para que los valores de estos restos sean adecuados hay que escoger cuidadosamente los valores del dividendo y del divisor. Una fórmula muy corriente es evaluar el resto de la siguiente división:

$$\frac{A * 899}{32767}$$

En *A* se coloca un número cualquiera. El resultado es un número comprendido entre 1 y 32765. Para calcular el siguiente número aleatorio se toma el anterior como nuevo valor de *A*. Este proceso se puede repetir tantas veces y se van obteniendo números al azar. No vamos a entrar ahora en detalles matemáticos, pero se obtienen un total de 16384 números distintos, pasados los cuales se empieza por el primero otra vez. Por tanto se trata de una serie de números. Es por esta razón que estos números reciben el nombre de *pseudo-aleatorios* ya que en realidad no se obtienen totalmente al azar.

Normalmente, los números entre 1 y 32765 se convierten en valores comprendidos entre 0 y 1 a base de dividirlos por 32767. Por otra parte, muchos ordenadores utilizan otros números distintos de 899 y 32767 a fin de que la secuencia de números aleatorios sea más larga y no se produzcan repeticiones hasta que no se hayan generado cientos de miles o incluso millones de números.

Además de usarse en la simulación, los números aleatorios se usan para los programas de juegos en los que conviene dotar a la máquina de un comportamiento imprevisible, de modo que el contrincante no pueda adivinar la estrategia que sigue el ordenador. La estadística es otro campo donde los números aleatorios tienen gran aplicación.

12.3.1 Generación números aleatorios

Función RND

Para generar números aleatorios, el BASIC dispone de una función especial. El nombre de esta función es: RND

Este nombre proviene de la abreviatura de la palabra inglesa «random» que significa «azar». No la incluimos en su momento en el catálogo de funciones puesto que ahora le reservamos varios apartados para explicar su funcionamiento.

Tal como indica su nombre, el resultado de esta función es numérico. Además es una función que carece de argumentos (este punto puede variar en algunas versiones del BASIC). El resultado es un número fraccionario comprendido entre 0 y 1, ambos exclusive, esto es, *sin incluir nunca* el 0 ni el 1, pero comprendidos entre ellos. Por la propia naturaleza de la función, el resultado es imprevisible. Escribamos

```
PRINT RND
```

En pantalla aparece un número. Si repetimos inmediatamente la misma instrucción anterior, el resultado será totalmente distinto. Una demostración más palpable de este hecho nos la proporciona el siguiente programa:

```
10 FOR I=1 TO 10
20   PRINT RND
30   NEXT I
```

Al efectuar un RUN, en pantalla aparecerá una lista de 10 números cuyos valores son imprevisibles, aunque todos ellos cumplen la norma de estar comprendidos entre 0 y 1. Por el momento ya sabemos obtener números al azar.

Observe que si ahora hace un NEW y vuelve a teclear el programa, los números obtenidos son los mismos que antes.

Por eso se denominan números pseudo-aleatorios porque aunque la serie es imprevista, si empezamos de nuevo se obtienen exactamente los mismos números.

Esta particularidad es importante para probar los programas pues nos permite obtener condiciones repetibles.

Para obtener una serie verdadera de números aleatorios existe una instrucción especial que estudiaremos en el párrafo siguiente.

Reducción del margen de azar en un intervalo

Sin embargo, para la mayoría de aplicaciones, estos números fraccionarios son poco útiles. En su lugar, nos interesaría mucho más disponer de números enteros comprendidos entre dos límites cualesquiera. No hay dificultad alguna para realizar esta conversión. Para ello emplearemos la función INT. Por ejemplo, si queremos obtener números enteros entre 1 y 100, escribiremos

```
PRINT INT(RND*100+1)
```

Al multiplicar la función por 100, obtendremos un número comprendido entre 0 y 100 ambos exclusive. Cuando le sumemos una unidad, el número

estará entre 1 y 101, pero todavía es un número fraccionario. Al aplicar la función INT, se suprimen los decimales, con lo que obtenemos valores enteros entre 1 y 100, incluyendo a ambos.

12.3.2 Cebado inicial

Como ya hemos mencionado, los números pseudo-aleatorios generados forman una serie. Ocurre que, en la mayoría de ordenadores, cuando se ejecuta el programa, la función RND empieza siempre desde la posición inicial de la serie, dando lugar a la misma secuencia de números. Por tanto, estos números dejarían de ser aleatorios ya que, cada vez que se ejecute el programa, obtendremos los mismos resultados.

Para evitar este problema, existe en BASIC una instrucción que sirve para escoger al azar la posición inicial de la serie. Esta instrucción es:

RANDOMIZE

Función de la instrucción RANDOMIZE

La palabra RANDOMIZE en castellano significa «convierte en aleatorio». Se coloca siempre antes de emplear la función RND.

Al ejecutarse esta instrucción se produce un cebado inicial de la secuencia aleatoria.

En algunas versiones del BASIC, no existe esta instrucción. En este caso, el ordenador realiza automáticamente el cebado inicial, es decir, efectúa un RANDOMIZE interno y por tanto no hace falta que el operador la utilice.

12.3.3 Aplicaciones

12.3.3.1 Tirada de un dado

El programa que veremos a continuación simula la tirada de un dado. Al tirar un dado, se puede obtener un resultado comprendido entre 1 y 6. Este programa imprimirá en pantalla una serie de números entre 1 y 6 distribuidos al azar, como si en realidad hubieran sido obtenidos por un dado.

El listado es el siguiente:

```
10 REM Tirada dado
20 RANDOMIZE
30 FOR I=1 TO 10
40   LET A=INT(RND*6+1)
50   PRINT A
60   NEXT I
```

Observemos que la primera instrucción (dejando a parte el REM inicial) es RANDOMIZE. A continuación, se establece un bucle para que se generen diez números. La línea 40 es la clave del programa. Siguiendo el procedimiento explicado anteriormente para transformar los números frac-

cionarios generados por RND en números enteros, multiplicamos por 6 y le sumamos 1. Seguidamente se aplica la función INT y el resultado se almacena en *A*. Con este procedimiento aseguramos que *A* contiene un número entero comprendido entre 1 y 6. La siguiente instrucción imprime el valor de *A*. Finalmente la línea 60 cierra el lazo.

Ejecutando varias veces el programa veremos que cada vez se genera una serie de números al azar. Además, cada serie es distinta de las demás.

12.3.3.2 Distribución estadística

Anteriormente hemos afirmado que los números aleatorios obtenidos mediante RND tienen todos la misma probabilidad de aparecer. En el programa anterior, todos los números comprendidos entre 1 y 6 tienen la misma probabilidad de ser generados. Si en lugar de 10 números, generamos muchos más y los contamos, veremos que aproximadamente habrá la misma cantidad de cada uno de ellos. Precisamente, el siguiente programa trata de este tema.

Si sumamos el valor de la tirada de dos dados simultáneamente, los resultados estarán comprendidos entre 2 y 12. Pero ahora la probabilidad no está igualmente repartida. Para que el resultado sea 2 sólo hay una posibilidad: que en ambos dados aparezca un 1. En cambio para que aparezca un 7 existen varias posibilidades: 4+3, 5+2, etc.

El programa simulará tiradas de una pareja de dados y las irá clasificando según el resultado. Al final, escribirá el resumen de tiradas donde se reflejará las diferencias de probabilidad.

El listado es el siguiente:

```
 10 REM Tiradas dos dados
 20 DIM T(11)
 30 FOR I=1 TO 50
 40    LET A=INT(RND*6+1)
 50    LET B=INT(RND*6+1)
 60    LET C=A+B
 70    LET T(C-1)=T(C-1)+1
 80    NEXT I
 90 FOR I=1 TO 11
100    PRINT I+1;TAB(5);T(I)
110    NEXT I
```

Utilizamos el conjunto *T* para llevar el recuento de cada uno de los 11 resultados posibles (de 2 a 12). El bucle formado por las líneas 30 y 80 genera 50 tiradas. En las líneas 40 y 50 se genera el resultado de cada dado. La variable C contiene el resultado, que estará comprendido entre 2 y 12. Entonces, para incrementar el contador correspondiente se utiliza la línea 70, en la que se calcula el elemento de *T* a base de restar 1 a *C*.

Cuando finaliza el bucle, en *T* están almacenados los recuentos de cada uno de los 11 resultados posibles.

A continuación se construye un lazo para escribir los resultados. Se imprimen dos columnas, una que indica la suma de los dos dados y la otra que refleja las veces que ha salido esa suma. Observaremos que son más frecuentes los resultados centrales (el 6, el 7 y el 8) que los extremos. Si en la línea 30, aumentamos el valor del bucle a 200, entonces la distribución es más cercana a la probabilidad real de que aparezca cada resultado concreto.

Como que este programa imprime un resultado en forma de tabla, podemos aprovechar la subrutina utilizada para dibujar barras. De este modo, tendremos una visión gráfica de los resultados.

Para ello cambiaremos el listado a partir de la línea 90, quedando

```
 90 LET M=T(1)
100 FOR I=2 TO 11
110    IF T(I)>M THEN LET M=T(I)
120    NEXT I
130 FOR I=1 TO 11
140    LET L=20*T(I)/M
150    GOSUB 500
160    PRINT I+1;TAB(5);B$
170    NEXT I
180 END
500 REM Dibujo Barras
510 LET B$=""
520 FOR B=1 TO L
530    LET B$=B$+"*"
540    NEXT B
550 RETURN
```

En las líneas que van de la 90 a la 120 se determina cuál es el elemento máximo de *T*. A continuación viene el bucle de escritura. Dentro de él, se pasa control a la subrutina 500 que construye la barra. Este segmento del programa es idéntico al ejemplo citado.

Al ejecutar este programa se imprimirán 11 barras de asteriscos (*) cuya longitud será proporcional al número de veces que ha salido cada resultado.

RESUMEN

Las pruebas de las subrutinas se pueden hacer en modo inmediato y es recomendable probar las subrutinas separadamente.

Es necesario tener muy presente cuándo los argumentos que se pasan a una subrutina son destruidos por el cálculo. En general, es recomendable utilizar algunas variables en las comunicaciones que no utilicemos nada más.

Cuando se quiere acceder a una subrutina se utiliza la instrucción LET para asignar los valores de los datos a las variables que sirven de argumento.

Un programa funciona tanto más rápido cuanto menos subrutinas tenga, pero esta ventaja se puede despreciar cuando el problema a resolver es suficientemente complejo.

Debe evitar la partición del programa en un excesivo número de módulos. No hay que caer en el extremo opuesto.

El equilibrio justo sólo se obtiene con experiencia, que debe adquirir realizando programas y leyendo programas que otros han realizado.

La realización de listados en un programa es una necesidad muy frecuente.

Un listado se divide en tres partes:

— Cabecera.

— Cuerpo.

— Pie.

Es adecuado programar cada una de estas partes como subrutinas en nuestro programa.

La manipulación de impresoras requiere enviar a veces caracteres de control, por ejemplo, para saltar página. Esto se realiza con la instrucción LPRINT CHR$ (<Carácter de control>). Los caracteres de control a enviar dependen de la impresora que se utilice.

Los números aleatorios son interesantes para hacer simulaciones de sucesos reales en los que interviene lo que vulgarmente llamamos suerte.

En primer lugar, los números aleatorios son contrarios a la naturaleza del ordenador.

El ordenador realiza siempre las mismas operaciones con una precisión rigurosa y de la misma manera.

Se han diseñado procedimientos que sirven para obtener series de números que cumplen bastante con las leyes del azar. Se trata de los números pseudo-aleatorios.

El mecanismo de sacar el resto y encadenar al siguiente es el más utilizado. Los números que se obtienen no verifican ninguna ley de orden que por esta razón se usan como aleatorios.

De hecho se trata de una secuencia que si se empieza con el mismo número se obtienen los mismos resultados. Aparece otra vez la precisión rigurosa del ordenador.

El BASIC utiliza la función RND para obtener un número comprendido entre 0 y 1 (sin incluir a ninguno de ellos).

El número obtenido es fraccionario, si se desea entero dentro de unos márgenes se utiliza el mecanismo siguiente:

INT (RND * N + 1),

en donde, N representa el número mayor que deseamos obtener.

La instrucción RANDOMIZE evita que cada vez que ejecutemos un programa realice la misma secuencia de números aleatorios.

Esta operación se denomina cebado inicial.

Cuando se prueba un programa es mejor no utilizar la instrucción RANDOMIZE pues si hay problemas sabemos exactamente que la situación inicial es la misma para cada prueba.

EJERCICIOS DE AUTOCOMPROBACION

Complete las frases siguientes:

21. Es necesario probar las subrutinas. Esto se puede hacer con la ejecución en modo ....

22. El paso de argumentos se hace con variables destinadas al efecto, que se inicializan antes de entrar en la subrutina mediante una instrucción de ....

23. Las subrutinas hacen que el programa sea más ....

24. Toda página de un listado se divide en cabecera, cuerpo y ....

25. La simulación se realiza en el ordenador mediante los números ....

26. La función intrínseca .... es la que tiene el BASIC para obtener números aleatorios.

27. Si iniciamos un programa siempre obtendremos la .... secuencia de números aleatorios. Esto es interesante para probar el programa.

28. La instrucción .... sirve para evitar que siempre se genere la misma secuencia de números aleatorios al iniciar un mismo programa.

29. La instrucción RND siempre nos devuelve un número comprendido en el intervalo 0 y 1 ambos ....

30. Para obtener un número aleatorio dentro de un margen determinado entero, se utiliza la función INT después de multiplicar el número aleatorio por el .... que queremos obtener y sumarle uno para evitar el cero.

Encierre en un círculo la letra que corresponda a la respuesta correcta.

31. Para obtener un número aleatorio entero comprendido entre 1 y 8, qué instrucción se debe utilizar:

a) LET A = INT (RND * 8)

b) LET A = RND * 8 + 1

c) LET A = INT (RND * 8 + 1)

d) LET A = INT (RND) * 8 + 1

32. Para obtener un número aleatorio fraccionario entre 1 y 9, ambos exclusive, qué instrucción se debe utilizar:

a) LET A = INT (RND * 8)

b) LET A = RND * 8 + 1

c) LET A = INT (RND * 8 + 1)

d) LET A = INT (RND) * 8 + 1

33. Para obtener un número aleatorio entero comprendido entre 10 y 20, ambos exclusive, qué instrucción debemos colocar:

a) LET A = INT (RND * 9 + 11)

b) LET A = INT (RND * 10 + 11)

c) LET A = INT (RND * 9 + 10)

d) LET A = INT (RND * 10 + 10)

34. Utilice la subrutina del algoritmo de Euclides para calcular el máximo común divisor de los números 18456 y 65536. El resultado es:

a) 3

b) 5

c) 8

d) 128

35. Utilice la subrutina del algoritmo de Euclides para calcular el máximo común divisor de los números 15625 y 4375.

a) 5

b) 13

c) 125

d) 625

36. Utilice la subrutina del algoritmo de Euclides para calcular el máximo común divisor de los números 28561 y 16807.

a) 1

b) 5

c) 7

d) 9763

37. Utilice la subrutina del algoritmo de Euclides para calcular el máximo común divisor de los números 12345 y 6789.

a) 1

b) 3

c) 12345

d) 6789

38. ¿Cuál de las subrutinas siguientes cuenta los blancos que hay en una cadena de caracteres?
Se utiliza la variable *C* para almacenar la cuenta y la variable *A$* es la que contiene la cadena cuando se llama a la subrutina.

a)
```
1000 REM Contar blancos
1010 LET L = LEN( A$)
1020 FOR I = 1 TO L
1030 IF MID$(A$,I,1) = " " THEN LET C=C+1
1040 NEXT I
1050 RETURN
```

b)
```
1000 REM Contar blancos
1010 LET C = 0 : LET L = LEN( A$)
1020 FOR I = 1 TO L
1030 IF MID$(A$,I,1) = " " THEN LET C=C+1
1040 NEXT I
1050 RETURN
```

c)

```
1000 REM Contar blancos
1010 LET C =.0 : LET L = LEN( A$)
1020 FOR I = 1 TO L
1030 IF MID$(A$,I,1) = " " THEN LET C= L-I
1040 NEXT I
1050 RETURN
```

d)

```
1000 REM Contar blancos
1010 LET C = 0 : LET L = LEN( A$)
1020 FOR I = 1 TO L
1030 IF LEFT$(A$,I) = " " THEN LET C=C+1
1040 NEXT I
1050 RETURN
```

39. ¿Cuál de las subrutinas siguientes cuenta el número de palabras que hay en una cadena de caracteres?
Se entiende por palabras cualquier secuencia de caracteres separados por blancos.
Por ejemplo, la cadena «LET A= 3*5 : REM Asignación» contiene las palabras: LET, A=, 3*5, :, REM y Asignación. Observe que es un concepto de palabra en un sentido muy amplio.
Se utiliza la variable *C* para almacenar la cuenta y la variable *A$* es la que contiene la cadena cuando se llama a la subrutina.
La variable *S* se utiliza para saber si el carácter anterior ha sido distinto de blanco. Cuando vale 0 es que el carácter anterior ha sido distinto de blanco y cuando vale 1 es que ha sido blanco.
Se empieza con *S* igual a 1, pues hay que suponer que el carácter anterior al inicial ha sido blanco.

a)

```
1000 REM Contar palabras
1010 LET C = 0 : LET L = LEN( A$): LET S=1
1020 FOR I = 1 TO L
1030 IF S<>0 THEN GO TO 1100
1040 IF MID$(A$,I,1) <> " " THEN GO TO 1200
1050 LET C = C+1 : LET S=1
1060 GO TO 1200
1100 IF MID$(A$,I,1) = " " THEN GO TO 1200
1110 LET S = 1
1200 NEXT I
1210 IF S = 0 THEN LET C = C+1
1220 RETURN
```

b)

```
1000 REM Contar palabras
1010 LET C = 0 : LET L = LEN( A$): LET S=1
1020 FOR I = 1 TO L
1030 IF S<>0 THEN GO TO 1100
1040 IF MID$(A$,I,1) <> " " THEN GO TO 1200
1050 LET C = C+1 : LET S=1
1060 GO TO 1200
1100 IF MID$(A$,I,1) = " " THEN GO TO 1100
1110 LET S = 0
1200 NEXT I
1210 IF S = 0 THEN LET C = C+1
1220 RETURN
```

c)

```
1000 REM Contar palabras
1010 LET C = 0 : LET L = LEN( A$): LET S=1
1020 FOR I = 1 TO L
1030 IF S<>0 THEN GO TO 1100
1040 IF MID$(A$,I,1) <> " " THEN GO TO 1200
1050 LET C = C+1 : LET S=1
1060 GO TO 1200
1100 IF MID$(A$,I,1) = " " THEN GO TO 1200
1110 LET S = 0
1200 NEXT I
1210 IF S = 0 THEN LET C = C+1
1220 RETURN
```

d)

```
1000 REM Contar palabras
1010 LET C = 0 : LET L = LEN( A$): LET S=1
1020 FOR I = 1 TO L
1030 IF S<>0 THEN GO TO 1100
1040 IF MID$(A$,I,1) <> " " THEN GO TO 1200
1050 LET C = C+1 : LET S=1
1060 GO TO 1200
1100 IF MID$(A$,I,1) = " " THEN GO TO 1200
1110 LET S = 0
1200 PRINT I
1210 IF S = 0 THEN LET C = C+1
1220 RETURN
```



SOLUCIONES DE LOS EJERCICIOS DE AUTOCOMPROBACION

Capítulo 9

1. Bucle.
2. Variable de control.
3. Instrucciones.
4. FOR, NEXT.
5. Inicio.
6. Final.
7. NEXT.
8. Expresión.
9. Incrementar.
10. Decreciente.
11. b
12. b
13. a
14. d
15. c
16. d
17. b
18. c
19. b
20. c
21. Cerrar.
22. Instrucciones.
23. Esperar.
24. GOTO.
25. FOR.
26. Variable de control.
27. Expresiones.
28. Entrar.
29. Cero.
30. Cerrar.
31. *a)* 31.7
    b) 31.2
    c) 31.9
    d) 31.6
    e) 31.8
    f) 31.4
    g) 31.5
    h) 31.10
    i) 31.1
    j) 31.3

Capítulo 10

1. tipo
2. conjunto
3. dato
4. una, vector
5. dos, matriz
6. dimensiones
7. números
8. restricción
9. dólar
10. antes
11. inicio
12. subíndice
13. paréntesis
14. comas
15. subíndice
16. F
17. V
18. F
19. V
20. F
21. b
22. d
23. c
24. d

25. c
26. modificación
27. FOR/NEXT
28. control
29. dimensiones
30. nombre
31. b
32. b
33. b
34. d
35. d

Capítulo 11

1. Fijos
2. DATA
3. Mezclarse
4. Datos
5. Situación
6. Orden
7. READ
8. Comas
9. Leído
10. RESTORE
11. a
12. c
13. d
14. b
15. d
16. b
17. c
18. b (y escribe los números 5,4,3,2,1)
19. b (y escribe los números 5,4)
20. c
21. Final
22. Funciones
23. FN
24. Textual
25. DEF
26. Argumentos
27. Propias
28. Coincidir
29. Variables
30. Situación
31. c
32. b
33. d, no coincide el orden de los argumentos
34. b
35. b
36. a
37. c
38. c
39. d, el argumento debe ser numérico
40. d, hay demasiados argumentos

Capítulo 12

1. subrutina.
2. memorice.
3. instrucción.
4. RETURN.
5. error.
6. invisible.
7. más.
8. descomponer.
9. argumentos.
10. funciones.
11. S y T.
12. S.
13. T.
14. S.
15. Ninguna.
16. a.
17. c.
18. d.
19. b.
20. d.
21. inmediato.
22. asignación.
23. lento.
24. pie.
25. aleatorios o seudo-aleatorios.
26. RND.
27. misma.
28. RANDOMIZE.
29. exclusive.
30. número o número máximo.
31. c.
32. b.
33. a.
34. c.
35. d.
36. a.
37. b.
38. b (La alternativa *a* no pone a cero la variable C. La alternativa *c* hace un cálculo equivocado en la línea 1030, LET C = L−I. La alternativa d utiliza la función LEFT$ en lugar de la MID$.)
39. c (La alternativa *a* no coloca a cero el valor de S cuando encuentra un carácter en blanco. (Línea 1110 LET S = 1.) La alternativa *b*, cuando se encuentra un carácter blanco se queda atrapada en la línea 1100 pues se recicla sobre sí misma. La alternativa *d* es incorrecta porque tiene un PRINT I en lugar de un NEXT en la línea 1200.)

Parte II

PRACTICAS CON EL ORDENADOR

Capítulo 9

ESQUEMA DE CONTENIDO

- Objetivos
- La estructura de repetición
- Instrucciones FOR y NEXT
- Bucle escalonado
- Bucles de espera
- Posibilidades de los bucles FOR/NEXT
- Práctica 1. El fondo cuadriculado
  - Escritura del problema de pantalla
  - Definición del programa
    - Variables
    - Líneas verticales
    - La entrada de datos
  - Pruebas
  - Observaciones finales
- Práctica 2. Construcción de una tabla financiera
  - Definición del problema
  - Escritura del programa

9.1 LA ESTRUCTURA DE REPETICION

En el programa que se da a continuación se pretende escribir una tabla de raíces cuadrados desde el 1 al 10.

```
10 LET vuelta = 1
20 IF vuelta > 10 THEN GOTO 9999
30     PRINT vuelta,SQR(vuelta)
40     LET vuelta = vuelta +1
50     GOTO 20
```

La decisión se toma al inicio del bucle en el ZX-Spectrum

Esta estructura de repetición es más adecuada para el ZX-SPECTRUM que la propuesta en el capítulo standar.

Las diferencias están en que la primera instrucción de la repetición es la pregunta de si el valor de la variable de control supera o no el valor final.

El cuerpo de la repetición acaba incrementando la variable de control y enviando el programa de nuevo a la pregunta que ha iniciado el bucle.

9.2 Instrucciones FOR y NEXT

La palabra FOR se encuentra sobre la tecla F. La palabra TO está escrita también sobre la tecla F. La instrucción NEXT se encuentra sobre la tecla N.

Como sabemos de los primeros capítulos, en el ZX-SPECTRUM los nombres de las variables de tipo numérico pueden tener cualquier número de letras. Sin embargo si una variable se va a utilizar como variable de control de un bucle, entonces su nombre está restringido a una sola letra.

En el programa del apartado anterior se ha empleado la variable *VUELTA* que nos sirve precisamente para contar el número de vueltas que da el programa. Si escribimos este mismo programa utilizando las instrucciones FOR/NEXT entonces no es posible usar la variable VUELTA puesto que tiene un nombre demasiado largo. En su lugar emplearemos,por ejemplo, la variable V, quedando

```
10 FOR V=1 TO 10
20     PRINT V,SQR(V)
30     NEXT V
```

La variable de control sólo debe tener una letra

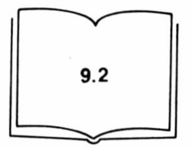
Como se aprecia, las instrucciones FOR/NEXT nos permiten programar con gran simplicidad un proceso repetitivo. Basta comparar la sencillez de este programa con el anterior. Recordemos también que en el ZX-SPECTRUM, la *variable de control debe tener un nombre formado por una sola letra.* Este hecho hay que tenerlo presente si se adapta un programa escrito en otro ordenador.

Como aprenderá en el capítulo standar, cuando el valor de finalización es más bajo que el valor inicial el comportamiento de las instrucciones FOR y NEXT depende del ordenador. En el caso del ZX-SPECTRUM, la pregunta se sitúa en la instrucción FOR, al inicio del bucle. Por lo tanto, el cuerpo del bucle no se ejecutará ninguna vez.

Así, por ejemplo,

```
FOR I = 10 TO 1
```

no pasará ninguna vez por el cuerpo del bucle.



9.3 BUCLE ESCALONADO

La palabra STEP se encuentra sobre la tecla D. A continuación de esta palabra se coloca una expresión numérica cuyo resultado puede ser positivo (bucle creciente) o negativo (bucle decreciente) o incluso fraccionario.

El siguiente programa es un ejemplo de bucle con incremento fraccionario.

```
10 FOR V=1 TO 3 STEP 0.5
20      PRINT V,SQR(V)
30      NEXT V
```

Se permite que el paso sea fraccionario

El valor de *V* empezará en 1 y se incrementará en 0.5 a cada vuelta dando 1.5, 2, 2.5 y 3.

En el caso de los bucles decrecientes debe utilizarse el valor del paso igual a –1.

Por ejemplo, el programa siguiente escribe en orden inverso los 10 primeros números.

```
10 FOR I = 10 TO 1 STEP -1
20 PRINT I
30 NEXT I
```

También en este caso, si el valor final es mayor que el inicial, el bucle no se ejecuta ninguna vez.

Así, la instrucción

```
FOR I = 1 TO 10 STEP -1
```

no permitirá alcanzar nunca el cuerpo del bucle.

9.4 BUCLES DE ESPERA

En el ZX-SPECTRUM, para conseguir un tiempo de espera de aproximadamente 1 segundo hay que emplear el siguiente bucle:

```
FOR I=1 TO 100 : NEXT I
```

Ciertamente, como ya sabemos, este ordenador incorpora una instrucción especialmente dedicada a esta función: la instrucción PAUSE.

Para esperar un segundo se utiliza la orden

```
PAUSE 50
```

La instrucción PAUSE puede sustituir a los bucles de espera

Esta instrucción es muy cómoda pero tiene el inconveniente de que no es compatible con muchas variantes del BASIC. Por el contrario, el bucle de espera funciona en todas las máquinas.

Quizá se pregunte por qué se debe utilizar 50. Pues porque la instrucción PAUSE utiliza para su contaje la frecuencia de la red de alimentación del ordenador que es de 50 ciclos por segundo.

Debe tener en cuenta que en varias partes del mundo se utilizan 60 ciclos por segundo como frecuencia de la red eléctrica, en este caso la instrucción conveniente es

```
PAUSE 60
```

9.7

9.5 POSIBILIDADES DE LOS BUCLES FOR/NEXT

El esquema general de las instrucciones FOR y NEXT en el ZX-SPECTRUM es:

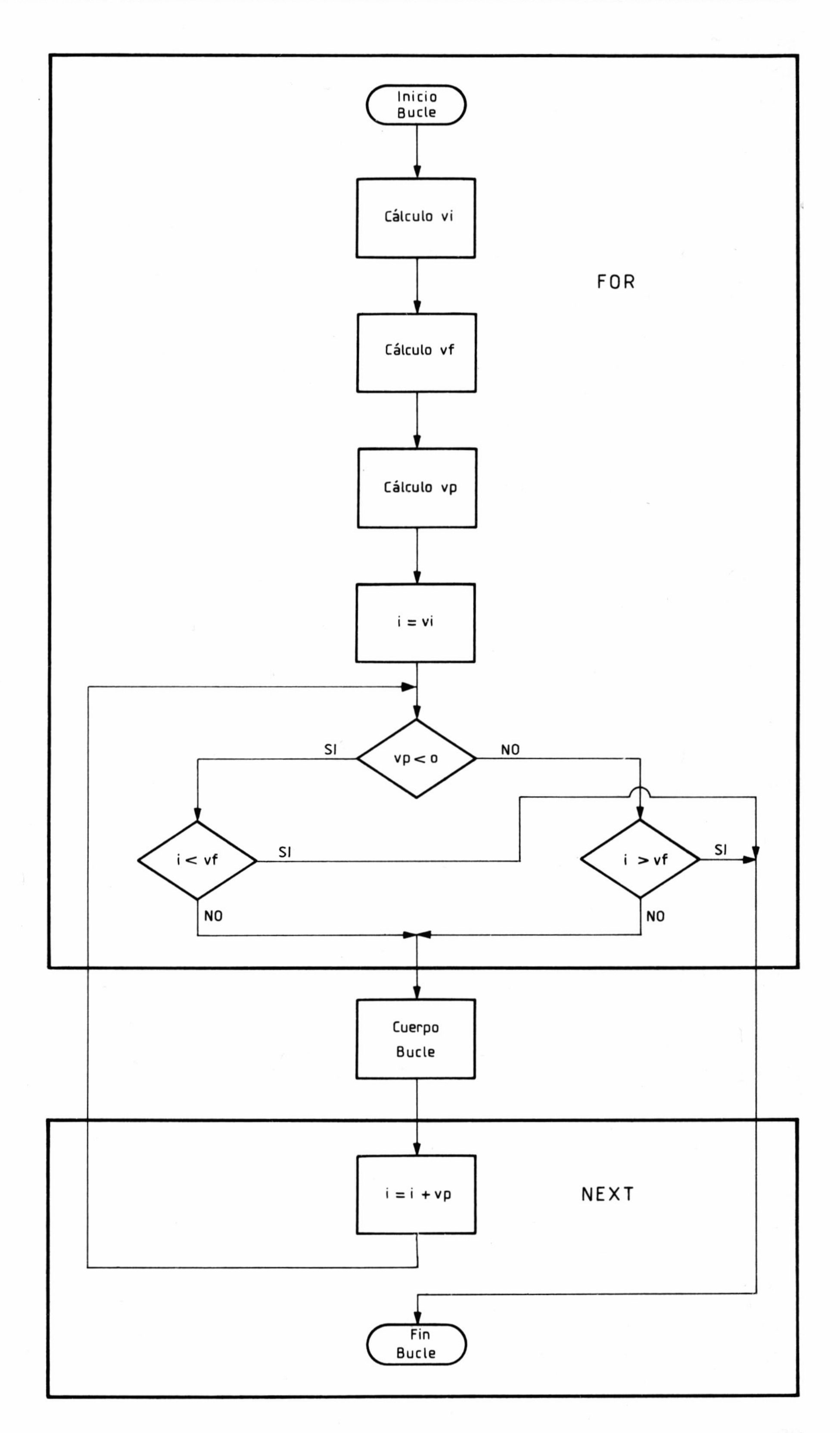
```
FOR i = vi TO vf STEP vp
    Cuerpo del bucle
    NEXT i
```

La variable de control es la *i*, y *vi*, *vf* y *vp* son abreviaturas para indicar las expresiones para calcular los valores inicial, final y el paso.

La figura 1 nos muestra el ordinograma de funcionamiento del bucle en el ZX-SPECTRUM. Vaya siguiéndolo.

Las observaciones que hay que hacer son las siguientes:

Figura 1 Ordinograma de funcionamiento de las instrucciones FOR y NEXT.



1. Se calculan primero las expresiones inicial (cálculo vi), final (cálculo vf) y paso (cálculo vp) y se almacenan como variables intermedias a las que el programador no tiene acceso. Este punto es importante para saber cómo funciona el ZX-SPECTRUM cuando en las expresiones de cálculo interviene la variable de control.

2. Se asigna a continuación el valor inicial a la variable de control (i = vi). Tenga en cuenta, por tanto, que si la variable de control interviene en las expresiones de los valores inicial (vi), final (vf) y paso (vp), el valor que se utiliza es el que tiene antes de alcanzar la instrucción FOR.

3. Se toma la decisión de continuar o no en el bucle (vp<0). Según que el paso sea positivo (salida hacia la derecha) o negativo (salida hacia la izquierda), la pregunta es distinta, como pone de manifiesto el ordinograma. Si el paso es negativo, se pregunta si la variable de control es menor que el valor final (i<vf). Si el paso es positivo se pregunta si la variable de control es mayor que el valor final (i>vf).

4. Se ejecuta el *cuerpo del bucle*. El número de instrucciones aquí es indeterminado y depende de las necesidades del programador. La variable de control puede ser alterada en este bloque pero como ya hemos indicado no es una práctica aconsejable.

5. Se incrementa la variable de control en el valor del paso (i=i+vp). No hace falta considerar ahora si el paso es positivo o negativo, pues la aritmética con números negativos funciona correctamente en el ordenador.

6. Se envía el control del programa a realizar de nuevo la pregunta, es decir el paso 3.

El funcionamiento completo de un bucle FOR/NEXT es complejo

Los pasos 1, 2 y 3 corresponden a la instrucción FOR. Los pasos 5 y 6 son los correspondientes a la instrucción NEXT. La complejidad de este ordinograma demuestra la comodidad que supone la utilización de las instrucciones FOR y NEXT, en lugar de expresarlo en función de instrucciones elementales.

Veamos un ejemplo para acabar de entender las propiedades y limitaciones del FOR/NEXT en el ZX-SPECTRUM.

Considere el programa siguiente:

```
20 FOR i = i+1 TO 2*i STEP i/2
30 PRINT i
40 NEXT i
```

El programa sólo tiene la finalidad de mostrar el comportamiento de un FOR cuando la variable de control interviene en el cálculo de las expresiones del valor inicial, final y paso.

El cuerpo del bucle es simplemente escribir la variable de control.

Si se ejecuta el programa tal como está se obtiene el error de

2 Variable not found, 20:1

es decir, que la variable no está definida. Esto ocurre porque primero se calculan las expresiones del valor inicial, final y paso. El valor de la i que se utiliza es justamente el que tiene la variable i antes de entrar en el FOR. Como realmente no está definida, nos da el mensaje de error.

Para hacer funcionar el programa se debe definir la variable i antes; por eso añadimos la instrucción

```
10 LET i=2
```

Al ejecutar ahora el programa se obtienen uno detrás de otro el 3 y el 4. La razón es que antes del bucle la i vale 2, el valor inicial es i +1, por lo tanto, vale 3. El valor final es 2*i, por lo tanto vale 4. El paso es i/2, por lo tanto vale 1. El bucle es equivalente a

```
FOR i=3 TO 4 STEP 1.
```

Cambie ahora el valor de la instrucción 10 a

```
10 LET i = 6
```

al ejecutar obtendremos 7 y 10. En efecto, ahora el inicio del bucle será el equivalente a:

```
FOR i=7 TO 12 STEP 3.
```

Finalmente coloque el valor de i a 5. Obtendrá los valores 6 y 8.5. La razón es que ha realizado un bucle con estos valores

```
FOR i=6 TO 10 STEP 2.5
```

Repetimos que aunque conozca bien el funcionamiento del bucle es desaconsejable utilizar este tipo de inicializaciones. Por una parte deberá recordar cada vez este funcionamiento y por otra puede ser que en otra máquina el comportamiento sea muy distinto.

9.6 PRÁCTICA 1. EL FONDO CUADRICULADO DE PANTALLA

Cuando queremos realizar un estudio sobre cualquier tema utilizamos las tablas.

Para realizar las tablas dibujamos unas líneas horizontales sobre el papel y unas líneas verticales. El resultado son varias casillas del tamaño necesario para la tabla que deseamos construir.

La figura 2 nos muestra distintas formas de tablas. El proceso de construcción es un proceso repetitivo típico, se tiran líneas paralelas en horizontal y luego líneas paralelas en vertical. Este trazado o dibujo se denomina soporte o fondo de la tabla.

El objetivo de esta práctica consiste en trazar líneas sobre la pantalla para construir el soporte de una tabla tal como lo hacemos en el papel.

Cuando decimos dibujar en el ordenador nos referimos a rellenar una línea de pantalla con algún carácter elegido previamente.

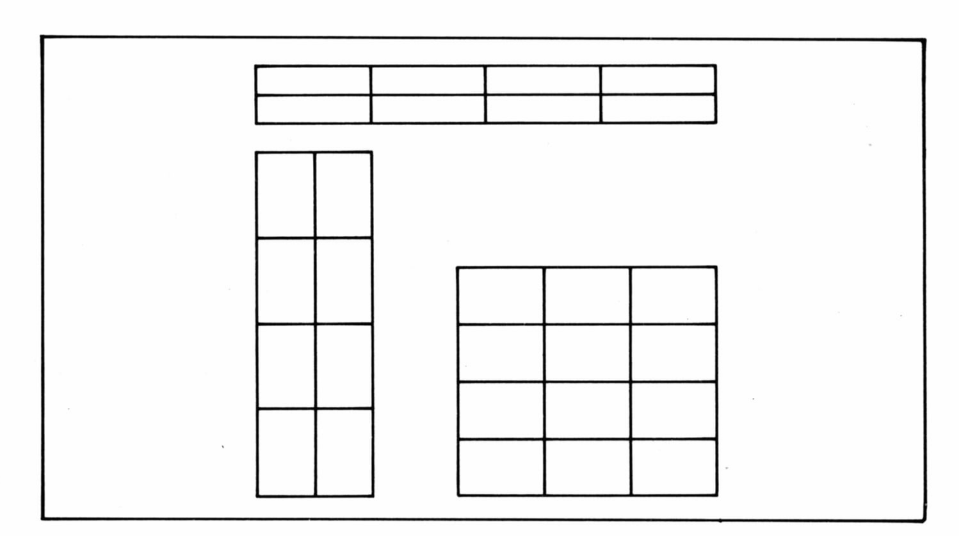


Figura 2 Soporte de tablas de diversas formas.

9.6.1 Definición del problema

Se trata de construir un programa para dibujar en la pantalla un soporte de tabla que tenga un número determinado de columnas, que abarque un número de caracteres fijado y que tenga un número determinado de filas cada una de las cuales abarque un número preestablecido de líneas de la pantalla.

Para aclarar el enunciado general del problema, consideremos un ejemplo concretando los valores de las distintas variables. Si deseamos una tabla de dos columnas que cada una tenga diez caracteres de ancho y cinco filas de dos líneas cada una, en la figura 3, se muestra cómo es la tabla sobre una plantilla de cuadrícula equivalente a la de los caracteres en pantalla.

Observe que sólo hemos utilizado asteriscos para delinear el soporte de la tabla. Podremos utilizar un símbolo distinto para los elementos verticales y horizontales tal como indica la figura 4.

Figura 3 Soporte de una tabla construida con el ordenador con un único símbolo.

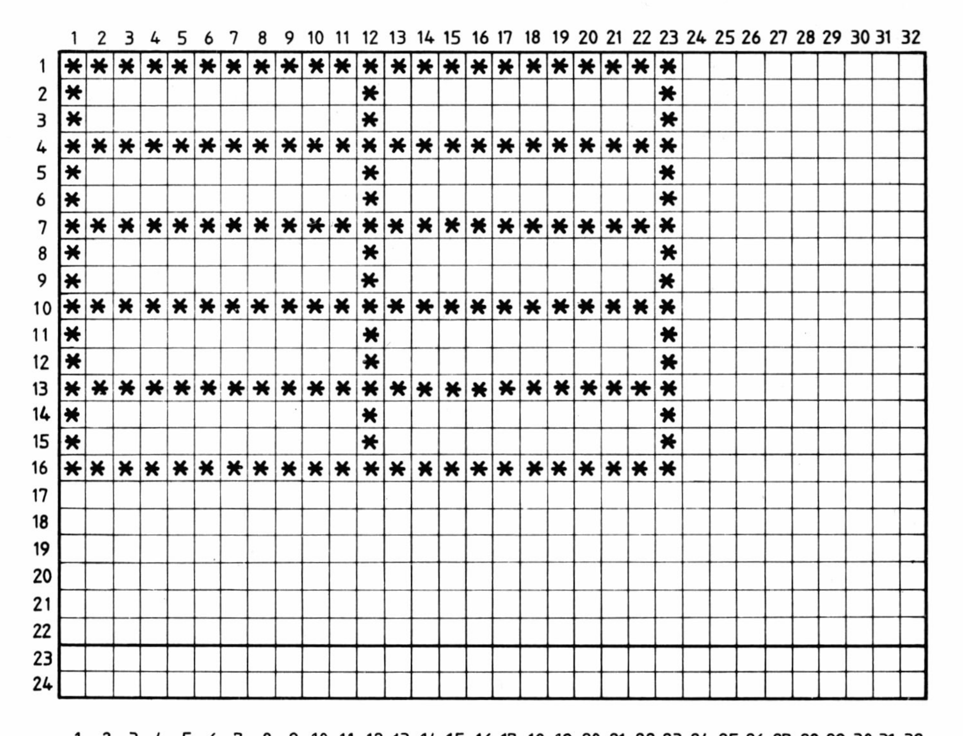
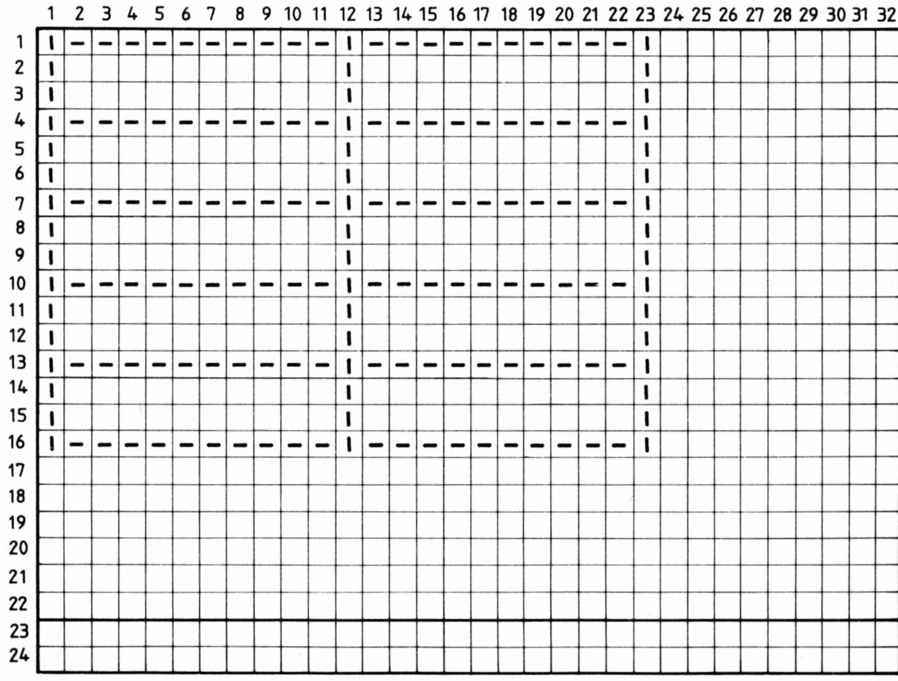


Figura 4 Soporte de tabla construida con ordenador con dos símbolos.



Las figuras 3 y 4 nos permiten saber cuántos datos precisamos para construir las tablas:

— Número de columnas que debe tener la tabla (en nuestro caso 2).

— Número de caracteres por columna (en nuestro caso 10).

— Número de filas (5 en el ejemplo).

— Número de líneas por fila (2 en el ejemplo).

— Carácter que sirve para denotar en espacio vertical. En la figura se utiliza la barra vertical (|).

— Carácter que sirve para denotar el carácter horizontal (en la figura se utiliza el guión (–).

Si consideramos otra vez la figura 4, está claro que el carácter de la línea 1 y la columna 5 debe ser un espacio horizontal. Sin embargo, en la línea 1 y la columna 1 no queda claro si hay que utilizar un espacio vertical u horizontal.

La decisión pertenece a cuestión de gusto y estética personal.

En esta práctica eligiremos el criterio de utilizar el símbolo de espacio vertical como prioritario al horizontal. Es decir, en todas las intersecciones entre líneas horizontales y verticales se colocara el símbolo vertical.

El problema y la decisión se pueden evitar si utilizamos el mismo símbolo para el espacio horizontal que vertical, como es el caso de la figura 3 en la que se ha elegido el asterisco.

9.6.2 Escritura del programa

9.6.2.1 Variables

A cada uno de los datos que hemos mencionado en el apartado anterior le asignaremos una variable. Además para empezar el diseño de la parte de dibujo tomaremos los valores que aparecen junto a las variables en la tabla siguiente:

c:	(2)	Número de columnas.
nc:	(10)	Caracteres por columna.
f:	(5)	Número de filas.
nf:	(2)	Número de líneas por fila.
v$:	(|)	Carácter de espacio vertical.
h$:	(–)	Carácter de espacio horizontal.

De momento dejaremos para más adelante el diseño de la entrada de datos y utilizaremos la asignación para obtener los valores de las variables del ejemplo que vamos a realizar.

Las instrucciones son las siguientes:

```
10 LET v$="!" : LET h$="-"
20 LET c = 2  : LET nc = 10
30 LET f = 5  : LET nf = 2
```

Consideremos en primer lugar la escritura de las líneas horizontales.

Debe haber un bucle que se repita f veces, es decir, el número de filas que debemos trazar. Cada una de estas repeticiones consiste en dibujar una línea (es equivalente a realizar un trazo en el papel con la regla) y saltarse a continuación tantas líneas como se requiera en forma de espacios en blanco (es equivalente a bajar la regla tantas líneas como queramos). Una vez finalizadas estas repeticiones se realiza una línea más. Tenga en cuenta que si la tabla tiene 5 filas debemos trazar 6 renglones sobre el papel, este renglón de más es el que trazamos después de finalizado el bucle.

El esquema es, por tanto,

FOR i = 1 TO f

Dibujar una línea horizontal

Saltarse tantas líneas como indique nf

NEXT i

Dibujar una línea horizontal

Para probar el funcionamiento correcto de este mecanismo vamos a escribir únicamente un carácter horizontal en las líneas llenas y una línea en blanco para las líneas a saltarse.

Tomemos como instrucciones provisionales las siguientes:

```
1000 FOR i = 1 TO f
1010    PRINT h$
1020    FOR j = 1 TO nf
1030       PRINT
1040       NEXT j
1050    NEXT i
1060 PRINT h$
```

Como puede ver, hay dos bucles anidados, el primero empieza en la instrucción 1000 y acaba en la instrucción 1050. Este bucle responde al mecanismo de repetición básico, esbozado en el esquema, para las filas que deseamos en nuestro caso 5.

El cuerpo del bucle consiste en:

— La línea 1010 que consiste en el dibujo de un carácter horizontal

que indica dónde debemos trazar una línea llena. En el papel un trazo con la regla,

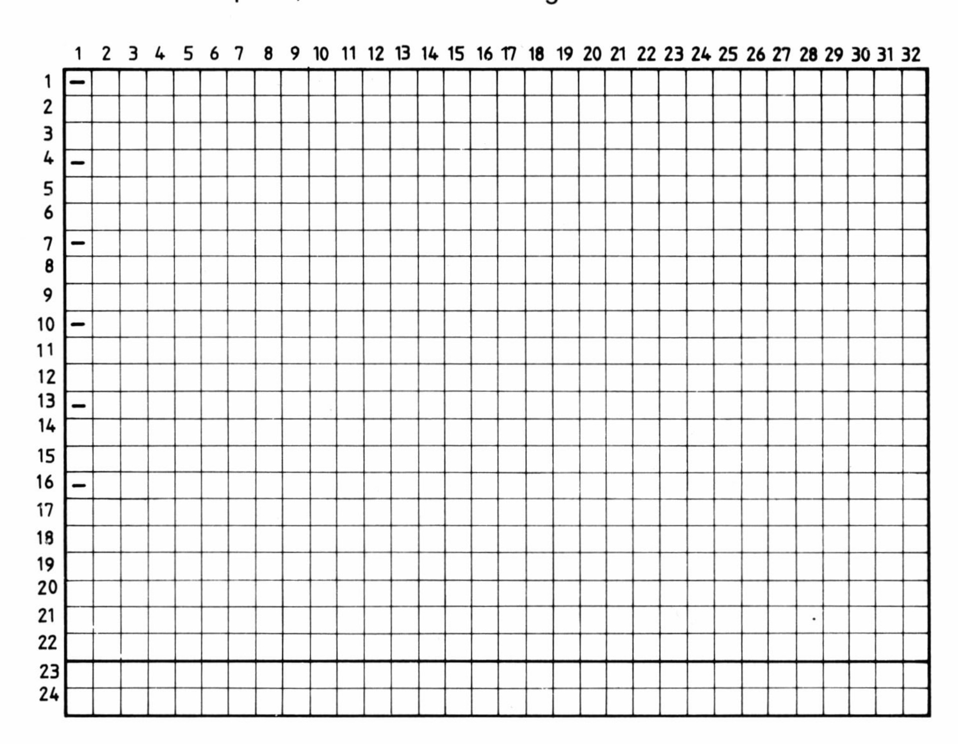
y

— El bucle desde la instrucción 1020 hasta la 1040 que responde al mecanismo de bajar la regla en el papel y consiste en imprimir líneas en blanco tantas como indica el dato nf, que en este ejemplo en particular vale 2.

Finalmente la instrucción 1060 escribe la línea adicional para cerrar la tabla por debajo.

Ahora teclee el RUN y ejecute el programa, en la pantalla aparecen unas rayas horizontales que indican dónde se deben dibujar las líneas horizontales del soporte, tal como indica la figura 5.

Figura 5 Construcción de una tabla. Disposición vertical.



9.6.2.2 Líneas verticales

Consideremos ahora la parte del problema asociada con el trazo de las líneas verticales. El mecanismo no es igual que el caso de las líneas horizontales ya que hay que trazar las líneas verticales a medida que trazamos las horizontales. Esta es una diferencia importante respecto al mecanismo de trazar tablas sobre un papel. Ciertamente si utilizamos la pantalla puede utilizarse la función AT y colocarnos en cualquier punto de ella. En el caso de utilizar la impresora este mecanismo no sirve.

En esta práctica consideraremos que la pantalla es equivalente a la impresora.

Hagamos el experimento siguiente con el programa que tenemos en el ordenador. Modifiquemos las instrucciones siguientes:

```
1010 PRINT v$;h$;h$;h$;v$
1030 PRINT v$;" ";" ";" ";v$
1060 PRINT v$;h$;h$;h$;v$
```

En la modificación de la instrucción 1030 se utiliza el blanco (" ") repetido tres veces para que quede más clara la semejanza con las otras modificaciones; evidentemente se pueden escribir tres blancos encerrados en unas solas comillas.

Ejecute el programa y observe que lo que aparece en pantalla debe ser como la figura 6.

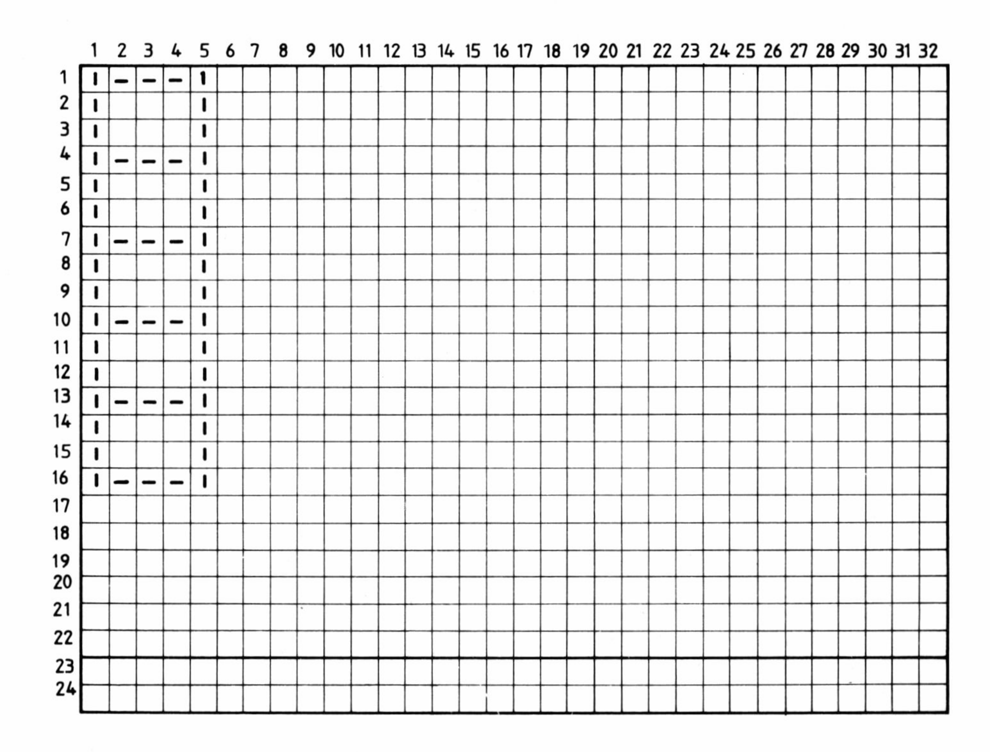


Figura 6 Pruebas de construcción de una tabla. Disposición horizontal.

Esta figura sugiere que el mecanismo para construir la tabla consiste en preparar previamente una línea de caracteres horizontales y en las columnas adecuadas los espacios verticales correspondientes.

También las líneas en blanco contienen algunos espacios verticales en las columnas precisas, que hay que preparar previamente a la escritura.

Llamemos a estas variables intermedias r$ (raya horizontal) y b$ (blanco). Para ilustrar el procedimiento modifiquemos de nuevo el programa, añadiendo las líneas 500 y 510 y modifiquemos las 1010, 1030 y 1060 para que el programa quede de la forma siguiente:

```
 500 LET r$=v$+h$+h$+h$+v$
 510 LET b$=v$+" "+" "+" "+v$
1000 FOR i = 1 TO f
1010    PRINT r$
1020    FOR j = 1 TO nf
1030       PRINT b$
1040       NEXTj
1050    NEXT i
1060 PRINT r$
```

Las instrucciones 500 y 510 utilizan la concatenación de cadenas de caracteres para construir las cadenas r$ y b$.

Se ejecuta el programa y el resultado es el mismo que el obtenido en la figura 6, es decir, el programa es equivalente al anterior.

La conclusión de este experimento es que el mecanismo de trazar rayas verticales consiste en preparar dos rayas horizontales, ambas con los caracteres verticales en el lugar oportuno, pero en una con caracteres horizontales en blanco y otra con caracteres horizontales el h$.

El mecanismo para preparar las rayas verticales es muy semejante al utilizado para trazar líneas en sentido horizontal.

El esquema es el siguiente:

texto a espacio vertical.

FOR i = 1 TO c

Añadir tantos caracteres de relleno como indica nc.

Añadir un espacio vertical.

NEXT i

Es decir, se parte de la cadena con un carácter vertical, se le añaden tantos caracteres de relleno como debe tener la columna (de blancos para la línea en blanco y de horizontales para la raya). Se añade a continuación un carácter vertical y se vuelve a iniciar el proceso hasta conseguir la anchura de los textos requerida.

Observe que en este caso también se añade un carácter vertical más que el que indica c, precisamente el que inicia las cadenas antes de iniciar el bucle.

La traducción de esta explicación al BASIC es la siguiente: (debe intentarla Ud. antes)

```
500 LET r$=v$ : LET b$= v$
510 FOR i = 1 TO c
520    FOR j= 1 TO nc
530       LET r$=r$+h$
540       LET b$=b$+" "
```

```
550      NEXT j
560     LET r$ = r$+v$
570     LET b$ = b$+v$
580     NEXT i
```

Al ejecutar ahora el programa debe resultarle una tabla de 2 columnas con 10 caracteres en cada columna y de 5 filas con dos líneas cada fila. Concuerda con la figura 4.

9.6.2.3 La entrada de datos

La última parte que nos queda por resolver es la entrada de datos. De momento utilizábamos la asignación para tener los datos necesarios, en las líneas 10, 20 y 30.

El primer impulso es colocar un INPUT para cada dato. Sin embargo hay que indicar una vez escogidos el carácter vertical y horizontal no desearemos modificarlo más que raras veces. En esta situación es mejor tocar el programa cuando queramos variar de símbolos.

Por lo tanto, sólo colocaremos los INPUT para los cuatro datos fundamentales, el número de columnas (c), caracteres por columna (nc), número de filas (f) y líneas por fila (nf).

También para que el programa se recicle añadiremos una instrucción al final para volver a pedir una entrada de datos.

Para completar este reciclaje, es necesario introducir un CLS para dejar la pantalla limpia entre tabla y tabla.

Las modificaciones que debemos hacer son:

```
  20 INPUT "Columnas:";c
  30 INPUT "Caracteres por columna:";nc
  40 INPUT "Filas:";f
  50 INPUT "Lineas por fila:";m
  60 CLS
2000 GOTO 20
```

Las cinco primeras instrucciones corresponden a la entrada de datos y la última al GOTO corresponde para hacer el reciclaje.

El programa completo es el siguiente:

```
10 LET v$="|":LET h$="-"
20 INPUT "Columnas:";c
30 INPUT "Caracteres por columna:";nc
40 INPUT "Filas:";f
50 INPUT "Lineas por fila:";nf
60 CLS
```

```
 500 LET r$=v$ : LET b$= v$
 510 FOR i = 1 TO c
 520   FOR j= 1 TO nc
 530     LET r$=r$+h$
 540     LET b$=b$+" "
 550     NEXT j
 560   LET r$ = r$+ v$
 570   LET b$ = b$+ v$
 580   NEXT i
1000 FOR i = 1 TO f
1010   PRINT r$
1020   FOR j = 1 TO nf
1030     PRINT b$
1040     NEXT j
1050   NEXT i
1060 PRINT r$
2000 GOTO 20
```

9.6.3 Pruebas

Hagamos con el programa las pruebas siguientes:

1. c=2, nc=10, f=5, nf=2

Debe reproducirnos la tabla de la figura 4.

2. c=10, nc=3, f=7, nf=3

Debido a que el número total de caracteres sobrepasa la anchura de la pantalla la tabla queda mal colocada.

El número de caracteres se puede calcular del modo siguiente: en cada columna hay 3 caracteres en blanco y uno de espacio vertical, en total 4 por columna, como hay diez columnas son 40. Además hay uno más, en total son 41.

Este hecho nos sugiere que en la entrada de datos podríamos calcular si el número de caracteres es mayor que la anchura de la pantalla e impedir que se realicen tablas que no van a caber.

Además como ahora escribimos muchas más líneas que las que caben en pantalla llegará un momento en que la máquina nos hará la pregunta scroll?

Esto también sugiere que en la entrada de datos deberíamos controlar el número de líneas a escribir, como en el caso de las columnas.

Observe que la anchura puede ser correcta y en cambio haber pedido un número de líneas excesivo. Por lo tanto, hay que calcular las dos posibilidades.

3. c=0, nc=5, f=3, nf=2

El punto interesante de esta prueba es que solicitamos cero columnas. Como el ZX-SPECTRUM sólo entra en el bucle si el límite inferior es menor o igual al límite superior, el programa no realiza ninguna vez el paso por el bucle que se inicia en la línea 510 y acaba en la línea 580. La raya y el blanco constan únicamente de un especio vertical.

La tabla consiste en 7 líneas con únicamente un carácter vertical cada una de ellas.

Observe que esto ocurrirá para cada valor de c inferior a 1, es decir, si coloca un número negativo a c, el efecto será el mismo que en el caso del cero.

4. c=5, nc=0, f=3, nf=2

El efecto es parecido al caso anterior, ahora no se entra en el bucle que se inicia en la línea 520 y finaliza en la línea 550.

Tanto la raya como el blanco quedarán con 6 caracteres verticales.

También aparecerán 7 líneas pues el número de filas y las líneas por fila son las mismas que antes.

5. c=2, nc=2, f=0, nf=3

Aquí la formación de r$ y b$ son normales.

La dificultad radica en que el número de filas es cero y, por lo tanto, nunca se entra en el bucle que empieza en la línea 1000 y acaba en la línea 1050.

El resultado será la impresión de una única raya, la que corresponde a la escritura en la línea 1060.

6. c=2, nc=5, f=3, nf=0

Aquí no se entra en el bucle que se inicia en la línea 1020 y acaba en 1040 pues nf es cero.

El resultado serán 4 rayas, 3 filas más la última para cerrar. En otras palabras, no habrá ninguna escritura de la variable b$.

Estas pruebas se han escogido para comprobar el funcionamiento del programa en situaciones límite. Las conclusiones son las siguientes:

— Si cualquiera de los datos es cero o negativo no tiene sentido hacer la tabla.

— Cuando el número de caracteres de r$ (o b$, es el mismo) supera la anchura de la pantalla, la tabla sale mal.

— Cuando el número de líneas supera a 22 la tabla sale mal.

9.6.4 Observaciones finales

Después de las conclusiones obtenidas de las pruebas se sugiere modificar la entrada de datos para evitar estos casos en que la tabla sale mal.

En primer lugar podemos aprovechar que no se admite cero o negativo en el número de las columnas para finalizar el programa (hasta este mo-

mento debíamos entrar un STOP en algún INPUT o mediante la tecla de interrupción si se estaba ejecutando el programa).

Como los demás datos tampoco pueden ser cero o negativos, si se entra un valor de este tipo puede servir para reiniciar la entrada de datos. Este sistema es útil pues uno puede darse cuenta que ha entrado un número de filas equivocado cuando el ordenador le pide el número de líneas por fila, se teclea en estos momentos un cero y se reinicia la entrada.

Para tener en cuenta este control se deben añadir las instrucciones de BASIC siguientes:

```
25 IF c <=0 THEN GOTO 9999
35 IF nc<=0 THEN GOTO 20
45 IF f <=0 THEN GOTO 20
55 IF nf<=0 THEN GOTO 20
```

La instrucción 25 sirve para finalizar y las demás para volver a empezar la entrada si nos equivocamos o si damos un valor incorrecto.

Por otra parte queremos limitar las tablas a la capacidad máxima de la pantalla. Es necesario, por lo tanto, hacer un cálculo previo para tener en cuenta si nos cabe o no en pantalla.

El número de columnas se calcula del modo siguiente:

$$(nc+1)*c+1$$

En efecto, cada columna tiene nc caracteres, además se le añade un espacio vertical, por eso le sumamos 1. Este cálculo corresponde al paréntesis (nc+1), como esto vale para c columnas el resultado lo multiplicamos por c, esto corresponde a (nc+1)×c, a esto le añadimos 1 pues es necesario cerrar con un carácter vertical.

Para el caso de las líneas la fórmula es muy parecida:

$$(nf+1)*f+1$$

el razonamiento idéntico al de las columnas.

Esto se aplica en el programa en las instrucciones siguientes:

```
100 LET i = (nc+1)*c+1
110 LET j = (nf+1)*f+1
120 IF ( i<= 32 AND j<=22 ) THEN GOTO 500
130 PRINT "La tabla no cabe en la pantalla"
140 GOTO 20
```

La sentencia 120 merece un comentario. En ella se toma la decisión de ir a construir la tabla. Esta decisión consiste en saber si se cumple que la anchura, representada por la variable i en estos momentos, es menor que 32, que es la anchura de la pantalla del ZX-SPECTRUM y si la longitud,

representada por j, es menor que la longitud de la pantalla del ZX-SPECTRUM. Para poder realizar la tabla deben cumplirse ambas condiciones a la vez; por lo tanto, las preguntas concretadas por los operadores relacionales menor o igual que (<=) se deben unir con un operador lógico AND, que sólo nos dará verdadero si ambas preguntas son verdaderas.

El programa completo es el siguiente:

```
  10 LET v$="|":LET h$="-"
  20 INPUT "Columnas:";c
  25 IF c <=0 THEN GOTO 9999
  30 INPUT "Caracteres por columna:";nc
  35 IF nc<=0 THEN GOTO 20
  40 INPUT "Filas:";f
  45 IF f <=0 THEN GOTO 20
  50 INPUT "Lineas por fila:";nf
  55 IF nf<=0 THEN GOTO 20
  60 CLS
 100 LET i = (nc+1)*c+1
 110 LET j = (nf+1)*f+1
 120 IF ( i<= 32 AND j<=22 ) THEN GOTO 500
 130 PRINT "La tabla no cabe en la pantalla"
 140 GOTO 20
 500 LET r$=v$ : LET b$= v$
 510 FOR i = 1 TO c
 520   FOR j= 1 TO nc
 530     LET r$=r$+h$
 540     LET b$=b$+" "
 550     NEXT j
 560   LET r$ = r$+ v$
 570   LET b$ = b$+ v$
 580   NEXT i
1000 FOR i = 1 TO f
1010   PRINT r$
1020   FOR j = 1 TO nf
1030     PRINT b$
1040     NEXT j
1050   NEXT i
1060 PRINT r$
2000 GOTO 20
```

9.7 PRÁCTICA 2. CONSTRUCCIÓN DE UNA TABLA FINANCIERA

Este programa pretende construir una tabla financiera según se muestra en la figura 7.

En la figura 7 tenemos en la dimensión vertical el número de años que depositamos en un banco una unidad monetaria.

Años	Interés		
	9	10	11
1	1.09000	1.10000	1.11000
2	1.18810	1.21000	1.23210
3	1.29502	1.33100	1.36763
4	1.41158	1.46410	1.51807
5	1.53862	1.61051	1.68505

Figura 7 Maqueta de una tabla financiera.

En el sentido horizontal diversos valores de rédito o interés, en nuestro caso 9, 10 y 11 % respectivamente.

En el interior de la tabla el valor que tendrá esta unidad monetaria si se deposita un número de años igual al que marca la fila en donde está situada y a un interés que marca la columna donde está situada. Por ejemplo, el valor 1.4641 es el valor que tendrá la unidad monetaria si la depositamos en un banco al 10 % y durante cuatro años. Está situada en la intersección del año número 4 y de la columna de interés del 10 %.

La fórmula para calcular los intereses compuestos es

valor = (1 + interés en tanto por uno) elevado al número de años

Es decir, debemos sumar 1 al interés dividido por 100, en el ejemplo que hemos puesto antes; 10 dividido por 100 es 0.1, a este número le sumamos 1, con lo que se obtiene 1.1. Este valor elevado al número de años, en nuestro caso 4, da 1.4641.

9.7.1 Definición del problema

Vamos a realizar un programa para el cálculo de una tabla financiera.

El número de años a considerar será de 1 a 5 fijo.

Las columnas del interés se calculan a partir de un dato que nos pide el programa más un 1 % más en las columnas sucesivas hasta llegar a tres columnas. Por ejemplo, si a este dato le damos el valor de 5, el programa construye la tabla para el interés de 5, 6 y 7 %.

Entre cada una de las filas de la tabla, el programa coloca una línea de guiones. Los valores del resultado serán de 7 cifras como máximo.

9.7.2 Escritura del programa

Las estructuras repetitivas en la tabla son fáciles de detectar.

El esquema puede ser el siguiente:

1. Inicializaciones.
2. Entrada del interés.

   Si el interés es cero o menor acabar.
3. Escribir la cabecera que consiste en:

   Escribir línea separación.

   Escribir títulos.

   Escribir los valores de los intereses.

   Escribir línea de separación.
4. Bucle para cinco años:

   Bucle para el cálculo del valor de cada columna.

Escribir los resultados de una fila.

Escribir la línea de separación.

5. Reciclar para pedir el nuevo interés.

Detallamos a continuación cada uno de los pasos:

1. Inicializaciones: Es preciso tener memorizada la línea de separación pues se utiliza muchas veces. Por lo tanto,

```
10 LET a$="--------------------------------"
```

De momento no se preocupe por el número de guiones a colocar, cuando tenga el programa casi finalizado los podrá ajustar mucho mejor.

2. Entrada de datos: No presenta ningún problema especial. Debido a que el rédito no puede ser cero o menor que cero se utiliza para finalizar el programa, las instrucciones son:

```
100 REM Entrada del interes
110 INPUT "Interes:";redito
120 IF redito <= 0 THEN GO TO 9999
```

3. Escritura de la cabecera.

Las instrucciones son las siguientes:

```
200 REM Escritura de la cabecera
210 CLS
220 PRINT a$:PRINT "Anyos     Interes"
230 FOR j = 0 TO 2
240 PRINT TAB(5+8*j);redito+j;
250 NEXT j
260 PRINT : PRINT a$
```

El único comentario es el bucle entre las líneas 230 a 250 que coloca los valores de los réditos a calcular, utilizando el tabulador que en el primer rédito vale 5, en el segundo vale 13 y en el tercero vale 21. El cálculo se realiza mediante la fórmula 5+8*j. Esta fórmula nos indica que el primero irá en la columna 5 cuando j vale cero. Entre un rédito y el siguiente dejará un espacio de 8 caracteres. El enunciado del problema nos dice que los números deben tener como máximo 7 caracteres, se deja uno más para dejar un espacio de separación entre los números.

Observe que el PRINT en el interior del bucle se acaba con un punto y coma pues no queremos mover el cursor.

Cuando se sale del bucle se coloca un PRINT sólo para que salte de línea y a continuación se escribe la línea de separación que finaliza la cabecera.

4. El bucle principal: Es la estructura repetitiva que nos escribe las líneas para cada año. La estructura es la siguiente:

```
500 REM Bucle principal
510 FOR i = 1 TO 5
900 NEXT i
```

Dejamos espacio suficiente para desarrollar todo el cuerpo del bucle.
La variable de control del bucle nos indica el número de años que estamos calculando.

5. Cuerpo del bucle. Las instrucciones son las siguientes:

```
600 REM Cuerpo del bucle
610 PRINT i;
620 FOR j = 0 TO 2
630 LET valor = (1+(redito+j)/100)^i
800 PRINT TAB(4+8*j);valor;
810 NEXT j
820 PRINT : PRINT a$
```

En primer lugar (línea 610) se escribe el número de años que está contenido en la variable i. Se coloca a continuación un punto y coma pues hay que continuar escribiendo en la misma línea.

De la línea 620 a 810 se realiza el bucle para los tres valores del rédito.

El cálculo del valor final se realiza en la línea 630, aplicando la fórmula que se menciona en el enunciado del problema. Observe que al rédito se le suma j para obtener el valor según en la columna que estamos. Este valor se divide por cien para expresarlo en tanto por uno. Se le suma 1 y se eleva a la potencia i que es justamente el número de años.

En la línea 800 se imprime el valor según indica el tabulador que sigue el mismo razonamiento que el explicado al escribir la cabecera. También se finaliza como punto y cosa para no ir más allá en la escritura de la línea.

En la línea 810 se finaliza el bucle y en la 820 se ejecuta un PRINT para ir a la línea siguiente (recuerde que se ha acabado la última impresión con punto y coma) y se imprime a continuación la raya de separación.

6. Reciclaje al inicio. Es simplemente una transferencia a la línea 100.

```
1000 GO TO 100
```

Ejecutamos ahora el programa. Dé como interés, por ejemplo, 9 para reproducir la figura 7.

Observará que cuando los números tienen muchas cifras decimales la construcción sale mal.

De hecho en todo el diseño hemos olvidado que nos dicen que sólo queremos 7 caracteres para los números.

Es necesario pasar el valor a cadena de caracteres antes de escribir y si sobrepasa los 7 caracteres cortarla. Lo podemos hacer con las instrucciones siguientes:

```
700 LET c$= STR$(valor)
710 IF LEN c$>7 THEN LET c$=c$(1 TO 7)
800 PRINT TAB(4+8*j);c$;
```

En la línea 700 pasamos *valor* a la cadena de caracteres c$ mediante la función STR$.

En la 710 preguntamos si la cadena sobrepasa los 7 caracteres, en este caso sólo tomamos los 7 primeros.

Finalmente debemos modificar la 800 para que nos escriba c$, en lugar de *valor*.

El programa completo es el siguiente:

```
  10 LET a$="------------------------------"
 100 REM Entrada del interes
 110 INPUT "Interes:";redito
 120 IF redito <= 0 THEN GO TO 9999
 200 REM Escritura de la cabecera
 210 CLS
 220 PRINT a$:PRINT "Anyos     Interes"
 230 FOR j = 0 TO 2
 240 PRINT TAB(5+8*j);redito+j;
 250 NEXT j
 260 PRINT : PRINT a$
 500 REM Bucle principal
 510 FOR i = 1 TO 5
 600 REM Cuerpo del bucle
 610 PRINT i;
 620 FOR j = 0 TO 2
 630 LET valor = (1+(redito+j)/100)^i
 700 LET c$= STR$(valor)
 710 IF LEN c$>7 THEN LET c$=c$(1 TO 7)
 800 PRINT TAB(4+8*j);c$;
 810 NEXT j
 820 PRINT : PRINT a$
 900 NEXT i
1000 GO TO 100
```

Las líneas en blanco que hay entre las líneas del programa es una manera de clarificar el listado, en la máquina no las podrá colocar.

Ahora es el momento de ajustar el número de guiones para tener una línea de las dimensiones necesarias.

Capítulo 10

ESQUEMA DE CONTENIDO

- Nomenclatura
- Instrucción DIM
- Variables textuales
- Extensiones de la instrucción DIM
- Empleo de la instrucción DIM
- Utilización de conjuntos dimensionados
- Elemento cero
- Procedimientos básicos
  - Llenar un conjunto
  - Escribir un conjunto
  - Localización por número
  - Localización por nombre
  - Fecha en letras
  - Cálculo de la media
- Práctica 1. El juego del Master Mind.
  - Reglas del juego
  - Definición del problema
  - Escritura del programa
    - La entrada de las combinaciones
    - El sistema de información
    - El cálculo de los marcadores
    - Encadenamiento de los ensayos
    - Obtención de la combinación oculta
    - Encadenamiento de partidas
  - Observaciones finales
- Práctica 2. Tabla de conversión de monedas
  - Objetivo
  - Escritura del programa
    - Tabla de conversiones
    - Mecanismo de pregunta
    - Escritura de los resultados
  - Pruebas

10.1 NOMENCLATURA

Los conjuntos dimensionados sólo se pueden definir con **una letra**

En el ZX-SPECTRUM, el nombre de los conjuntos dimensionados consta de una sola letra. Si el conjunto es de tipo textual, además de la letra, el nombre llevará el símbolo dólar ($) al final del mismo. Por tanto, no podemos manejar más de 26 conjuntos a la vez puesto que sólo disponemos de 26 letras para nombrarlos. Sin embargo, esta limitación carece de importancia, ya que muy pocas veces se trabaja con más de 4 ó 5 conjuntos a la vez.

10.2 INSTRUCCION DIM

La palabra DIM se encuentra sobre la letra D. En el ZX-SPECTRUM se admiten conjuntos de una, dos, tres o más dimensiones. Para las variables de tipo numérico, la instrucción DIM se comporta de forma estándar. Por el contrario, para las de tipo textual existen algunas peculiaridades que veremos a continuación.

10.3 VARIABLES TEXTUALES

Las variables textuales dimensionadas tienen todas el mismo número de caracteres

En el BASIC estándar cada elemento de un conjunto dimensionado puede tener un número de caracteres distinto de los demás elementos. Esto no es posible en el ZX-SPECTRUM. En este ordenador todos los elementos contienen exactamente el mismo número de caracteres. Este número hay que indicarlo en el momento de establecer la dimensión. La forma de indicarlo es añadiendo otro número a la lista de dimensiones. Este último número es el que indica cuántas letras o caracteres contendrá cada elemento del conjunto.

La instrucción

```
DIM N$(20,15)
```

significa que N$ es una lista textual de 20 elementos, cada uno de los cuales tiene 15 letras. Asimismo la instrucción

```
DIM T$(30,2,25)
```

establece que T$ es una tabla de 30 filas y dos columnas. Cada elemento de T$ contendrá 25 símbolos.

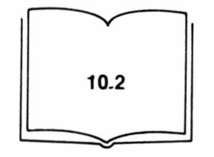
Si sólo indicamos una dimensión, por ejemplo:

```
DIM A$(15)
```

Limitación de longitud en las variables textuales

significa que limitamos la longitud total de A$, en lugar de establecer una lista. Es decir, que A$ continúa siendo una variable simple de tipo textual, pero que sólo puede contener 15 letras como máximo.

Este comportamiento se aparta notablemente del BASIC estándar. Convendrá pues, vigilar atentamente este punto cuando queramos realizar adaptaciones de programas escritos para otros ordenadores y viceversa. No olvidemos que esta diferencia de comportamiento sólo se aplica a conjuntos de tipo textual. Los numéricos se comportan de forma normal.



10.4 EXTENSIONES DE LA INSTRUCCION DIM

En el BASIC del ZX-SPECTRUM se admite la utilización de una expresión dentro de un DIM. Por tanto, es válido escribir

```
10 LET N=30
20 DIM A(N)
```

Cada conjunto dimensionado requiere expresamente una instrucción DIM

Sin embargo, en una sola instrucción DIM no se permite definir más de un conjunto. Por tanto, en el ZX-SPECTRUM no se admite una instrucción del tipo

```
10 DIM N$(10,25),A(25)
```

sino que habrá que utilizar dos instrucciones

```
10 DIM N$(10,25)
20 DIM A(25)
```

o bien, colocando las dos instrucciones en una sola línea

```
10 DIM N$(10,25) : DIM A(25)
```

10.5 EMPLEO DE LA INSTRUCCION DIM

Como ya sabemos, la instrucción DIM debe usarse antes de utilizar cualquier elemento de un conjunto. En el ZX-SPECTRUM al contrario del BASIC estándar sí que está permitido cambiar la dimensión de un conjunto durante la ejecución. Sin embargo, es una práctica que no recomendamos en absoluto, por dos razones:

La primera de ellas es que nuestros programas serán incompatibles con otros ordenadores.

La segunda razón es más importante. Ya hemos comentado en capítulos anteriores que cuando se elabora un programa para resolver un nuevo problema, la primera tarea a realizar antes de ponernos a teclear es efectuar un diseño previo. Una parte de este diseño consiste en construir una lista de variables y conjunto con sus dimensiones. Por tanto, si se establece el tamaño de un conjunto al principio, no tiene sentido retocarlo a medio programa.

La dimensión máxima utilizable depende del modelo (16K o 48K) y del gasto de memoria efectuado por el resto del programa. Si se emplea una dimensión demasiado grande, como por ejemplo

```
DIM A(9000)
```

el ordenador da el siguiente mensaje

4 Out of memory 0:1

que significa que el ordenador no tiene espacio suficiente en la memoria para almacenar el conjunto A.

10.6 UTILIZACION DE CONJUNTOS DIMENSIONADOS

En la utilización de conjuntos dimensionados de tipo numérico no hay ninguna diferencia respecto al BASIC estándar. En cambio, para los de tipo textual existen algunas variaciones. Las diferencias vienen motivadas por la distinta forma que tiene el ZX-SPECTRUM de establecer las dimensiones.

Como sabemos, en el ZX-SPECTRUM todos los elementos del conjunto tienen exactamente el mismo número de símbolos. Como consecuencia, si intentamos almacenar un texto que tenga más caracteres que los admitidos, quedará truncado. Si por el contrario, se almacena un texto con menor número de símbolos, entonces el espacio sobrante se llena con espacios en blanco. Veámoslo con un ejemplo

```
10 DIM B$(4,6)
20 LET B$(1)="CASA"
30 LET B$(2)="CASTILLO"
```

El elemento textual se rellena con blancos hasta la longitud especificada en la dimensión

En la línea 10, se define una lista B$ de cuatro elementos, cada uno de los cuales tiene 6 letras. en la línea 20 asignamos la palabra CASA de 4 letras al primer elemento de la lista. Puesto que quedan dos posiciones por ocupar (6-4=2) se almacenan dos espacios en blanco al final de CASA. Es decir, que en lugar de almacenar «CASA» de longitud 4, se almacena «CASA » de longitud 6.

En la línea 30 ocurre lo contrario. La palabra CASTILLO tiene 8 caracteres. Por tanto quedará cortada y se almacenan únicamente los 6 primeros

caracteres. Es decir, que el segundo elemento de la lista contendrá «CASTIL».

Podemos comprobarlo escribiendo, después de efectuar un RUN, la instrucción

```
PRINT B$(1);B$(2)
```

que escribirá en pantalla

CASA CASTIL

De nuevo insistimos que este comportamiento se aparta notablemente del funcionamiento estándar. Convendrá pues vigilarlo, especialmente cuando utilicemos la instrucción IF/THEN. Al contener un número extra de espacios en blanco, el BASIC encuentra que dos textos son distintos cuando originalmente eran iguales. Para comprender mejor este punto añadiremos la siguiente instrucción al programa anterior

```
40 IF B$(1)<>"CASA" THEN PRINT "DISTINTOS"
```

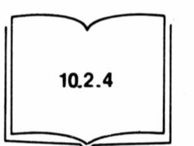
Al efectuar un RUN, el programa escribirá el mensaje «DISTINTOS». Esto ocurre porque «CASA» con dos espacios en blanco es distinto (desde el punto de vista del ordenador) de «CASA» sin espacios.

Este problema es típico del ZX-SPECTRUM y no aparece en el BASIC estándar. Sin embargo existen soluciones que veremos más adelante.

10.7 ELEMENTO CERO

No hay elementos con subíndice cero

En el ZX-SPECTRUM no existen los elementos con índice cero. Esto no constituye ningún problema puesto que generalmente no se trabaja con estos elementos



10.8 PROCEDIMIENTOS BASICOS

10.8.1 Llenar un conjunto

Los programas para llenar conjuntos numéricos no sufren variación respecto a lo que estudiará en la lección estándar. Por el contrario, en el primer programa utilizado para llenar el conjunto de tipo textual deberá cambiarse la línea 10 por

```
DIM B$(4,6)
```

Igualmente para el segundo ejemplo, la línea 10 queda

```
10 LET N=4 : DIM B$(N,6)
```

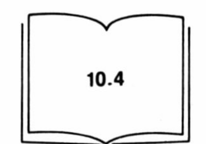
10.8.2 Escribir un conjunto

Como sabemos por las primeras lecciones, el ZX-SPECTRUM escribe los resultados numéricos sin añadirles espacios en blanco. A causa de esto, al escribir un conjunto de dos dimensiones en forma de tabla, las cifras de los elementos de una columna quedan pegadas a las de la siguiente columna.

Para evitarlo, en el programa que realiza esta operación se cambia la instrucción 100 por

```
100 PRINT C(I,J);" ";
```

De esta forma cada resultado quedará separado de los demás por un espacio en blanco



10.8.3 Localización por número

Al tratarse de un conjunto de tipo textual habrá que realizar en el programa estándar las modificaciones ya comentadas. Entonces la línea 20 queda:

```
20 LET N=5 : DIM A$(N,30)
```

Se toma el valor 30 para la longitud total de cada texto almacenado. Si se prevé que los textos serán más cortos o más largos se cambiará el 30 por el valor adecuado.

10.8.4 Localización por nombre

En este programa hay que introducir dos modificaciones para adaptarlo al ZX-SPECTRUM. La primera de ellas afecta a la instrucción DIM. La línea 20 queda

```
20 LET N=5 : DIM A$(N,10)
```

Reservamos 10 caracteres para cada elemento de la lista. Esto significa que si utilizamos nombres con menos letras, se completarán con es-

pacios en blanco. Entonces para que la línea 90 funcione correctamente deberemos entrar el nombre a localizar con los espacios en blanco correspondientes hasta completar 10 caracteres en total. Es obvio que con este procedimiento es fácil que el operador cometa errores al contar el número de espacios en blanco. Esto daría como consecuencia que el programa no localizaría nombres que en realidad sí están en la lista.

Este problema se puede solventar haciendo que en la instrucción IF de la línea 90 se compruebe la igualdad añadiendo al texto entrado por el operador, los espacios en blanco suficientes para que la longitud sea la misma que los elementos de la lista.

Se puede utilizar el valor 10 utilizado en la dimensión de tal manera que combinándolo con la función LEN se debe añadir la instrucción

```
85 IF LEN (X$)< 10 THEN LET X$=X$+" ": GO TO85
```

Hay que rellenar la entrada con blancos

Si se quiere hacer el programa independiente de este 10 ya que se puede modificar la dimensión y olvidarse de tocar esta instrucción. Se utiliza la función LEN para determinar la longitud del elemento de la tabla. La instrucción queda entonces como

```
85 IF LEN(X$) < LEN(A$(I)) THEN LET X$=X$+" ":GO TO 85
```

También como todos los elementos de A$ (I) tienen la misma longitud, es posible añadir estos blancos antes del bucle y ahorrarnos la repetición de esta operación cada vez que repetimos la pregunta en el bucle. Así se puede escribir

```
75 IF LEN(X$) < LEN(A$(1)) THEN LET X$=X$+" ": GO TO 75
```

Este ejemplo no enseña que hay que evitar colocar en el interior de los bucles operaciones innecesarios pues el tiempo de cálculo se alarga notablemente debido al esquema repetitivo del bucle.

Observe que la primera solución se obtiene aplicando un criterio rápido, cuando reflexionamos más profundamente encontramos el punto justo de programación: como todos los elementos de la tabla tienen la misma longitud se puede alargar X$ antes de entrar en el bucle.

Este procedimiento o variantes del mismo es el que utilizaremos en el ZX-SPECTRUM cuando los espacios en blanco de relleno nos causen dificultades. Tenga en cuenta estas observaciones al estudiar ahora la lección estándar.

10.8.5 Fecha en letras

Como de costumbre, modificaremos la línea que contiene la instrucción

DIM. El mes que tiene mayor número de letras es septiembre con 10. Por tanto, la lista N$ tendrá 12 elementos de diez letras. La línea 20 queda

```
20 DIM N$(12,10)
```

Como ya dijimos en la lección dedicada a funciones, en el ZX-SPECTRUM no existen las funciones estándar LEFT, MID y RIGHT. En su lugar se usa el operador de fragmentación. Las líneas 170, 180 y 190 quedan

```
170 LET D$=F$(1 TO 2)
180 LET M$=F$(4 TO 5)
190 LET A$=F$(7 TO 8)
```

El resto del programa no sufre variación.

10.8.6 Cálculo de la media

Puesto que se trabaja con conjuntos numéricos, no hay que introducir modificaciones en los programas y funcionan directamente en el ZX-SPECTRUM.



10.9 PRACTICA 1. EL JUEGO DEL MASTER-MIND

10.9.1 Reglas del juego

El juego del Master-Mind consiste en oponer a dos jugadores. El primer jugador, que llamamos A, dispone de seis peones de colores distintos. De estos peones escoge cuatro que sitúa en un orden determinado dentro de un cuadro preparado al efecto.

Por ejemplo, los colores de los seis peones son: rojo, azul, verde, amarillo, naranja y marrón. Elige 4 peones el rojo, azul, verde, amarillo y los coloca en este orden en un cuadro de cuatro casillas. Es importante mantener el orden, es decir, en el primer cuadro se coloca el rojo, en el segundo cuadro se coloca el azul, en el tecer cuadro se coloca el verde y finalmente en el cuarto cuadro se coloca el amarillo.

Esta ordenación de peones en el cuadro se mantiene oculta al segundo jugador.

Observe que esta elección de cuatro colores entre los seis que hay impide que en la configuración oculta se repitan los colores de un juego a otro.

El segundo jugador, que llamamos B, intenta acertar la combinación oculta que ha realizado A mediante ensayos sucesivos.

Después de realizar un ensayo el jugador A informa al B de una manera limitada. La información que suministra A a B es:

a) Número de peones que ha acertado tanto en color como en posición. Se utilizan para esta tarea marcadores de color negro.

b) Número de peones que ha acertado en color pero que están en una posición incorrecta. Se utilizan entonces los marcadores blancos.

Veamos un ejemplo, si la combinación oculta es:

rojo azul verde amarillo

y el jugador B da como combinación de prueba:

rojo verde marrón naranja

La información que debe suministrar A a B es de un marcador negro y uno blanco. El marcador negro proviene de que el peón rojo se ha acertado en color y posición. El marcador blanco proviene de que el peón verde se ha acertado en color pero no en posición.

Con esta información el jugador B ensaya una nueva combinación que le aporta una nueva información y así sucesivamente hasta que consiga acertar la combinación.

10.9.2 Definición del problema

El objetivo de esta práctica consiste en realizar un programa que actúe como jugador A, siendo el B el operador que actúa sobre el teclado.

Por otra parte, en lugar de utilizar los colores se utilizan las seis primeras cifras arábigas, es decir, el 1, 2, 3, 4, 5 y 6. Este tipo de codificación es más conveniente al ordenador y no introduce ninguna modificación esencial al juego. Cada cifra es el código de un color.

El problema se puede descomponer en cinco partes:

1. Hay que comunicarse con el exterior para pedir la combinación que ensaya el jugador B, manteniendo un sistema de información de los ensayos realizados.

2. Dada una respuesta de B el programa debe determinar la contestación de los marcadores.

3. Hay que definir la configuración oculta.

4. Encadenar un ensayo después de otro en tanto la respuesta de los marcadores no sea cuatro negro, que significa que se han acertado los cuatro colores en las cuatro posiciones; en este caso el jugador B ha ganado y no se ha sobrepasado un número determinado de ensayos. Si se sobrepasa el número de ensayos sin acertar los cuatro colores, el jugador B ha perdido.

5. Mecanismo de encadenar un juego detrás de otro.

Respecto a los datos, utilizaremos una lista de las cifras de tal manera que el contenido es la cifra que está en una determinada posición. Por ejem-

plo, a la combinación oculta le asignaremos una lista (conjunto dimensionado de una dimensión) de cuatro elementos que llamaremos *r* que contiene en cada elemento la cifra que está en la posición de la lista. Así,

r(1)=3;r(2)=6;r(3)=2;r(4)=1

indica que la cifra 3 está en primera posición, la cifra 6 está en segunda posición, la cifra 2 está en tercera posición y la cifra 1 está en cuarta posición.

De la misma manera la combinación de ensayo se almacena para el análisis de la información a suministrar a B en otra lista que tiene las mismas características y que se denomina *a*.

Esto obliga a definir con la instrucción DIM cada una de estas listas y es la primera instrucción del programa.

10.9.3 Escritura del programa

Según las definiciones de los datos la primera instrucción del programa debe ser dimensionar la variable que contendrá la combinación oculta y la variable que contendrá la combinación que le da el operador para tratar de averiguarla. Por tanto, escribiremos:

```
10 DIM r(4):DIM a(4)
```

Recordemos que *r* es la combinación oculta y *a* es la combinación de ensayo (deben utilizarse dos instrucciones DIM en el ZX-SPECTRUM).

Para simplificar el problema considere que la configuración oculta es la que se ha dado antes, es decir,

r(1)=3;r(2)=6;r(3)=2;r(4)=1

La generación de esta configuración oculta la diferimos hasta que se resuelva el resto del programa, por lo tanto, provisionalmente escribimos las instrucciones:

```
 10 DIM r(4) : DIM a(4)
100 LET r(1)= 3 : LET r(2) = 6
110 LET r(3)= 2 : LET r(4) = 1
```

Reservamos el bloque de instrucciones de la 100 hasta la 500 para la programación de la elección de la combinación oculta.

10.9.3.1 La entrada de las combinaciones

El segundo paso consiste en realizar la entrada de datos por parte del jugador B.

Lo único que hay que controlar es que los números que se entran estén comprendidos entre 1 y 6. Observe que no hay que controlar si se repiten las cifras o no. Hay jugadores que utilizan la estrategia de repetir números para mejorar la información.

El esquema de comprobación es:

```
500 INPUT a
510 IF 0<a AND a<7 THEN GOTO 550
520 PRINT "Cifra incorrecta"
530 GOTO 500
550 ...seguir.
```

La línea 510 es una pregunta para decidir si el número entrado está comprendido entre 1 y 6 ambos inclusive. La técnica se ha explicado ampliamente en la lección 6.

Este trozo está pensado para entrar en una sola cifra, pero el problema se repite para las cuatro cifras, basta sustituir *a* por los elementos de la lista. Tal como está la utilización de *a* es incorrecta pues se trata de una variable dimensionada. (De hecho el ZX-SPECTRUM no le señala error, es decir, puede utilizar el mismo nombre de variable para una dimensionada y otra sin dimensionar. Es una práctica que no le recomendamos, porque se presta a mucha confusión y hace el programa inadaptable a otros BASIC).

Es necesario añadir un bucle además para que se produzca la entrada de datos de cada una de las cifras. Un esquema de esta entrada es el siguiente:

```
500 FOR I = 1 TO 4
510 INPUT "Elemento "+STR$(i)+":";a(i)
520 IF 0<a(i) AND a(i)<7 THEN GO TO 550
530 PRINT "Cifra incorrecta"
540 GO TO 510
550 NEXT i
```

Esta versión es esencialmente la misma, lo único que se ha añadido es una indicación del elemento que estamos entrando en el INPUT y el bucle para entrar las cuatro cifras.

Sin embargo, si pensamos en la mecánica de entrar nos daremos cuenta que cuando estamos entrando la tercera cifra nos gustaría modificar la primera. Es decir, en cualquier momento de la entrada debemos prever que se puede anular y reiniciar el proceso. Para realizar este proceso se puede utilizar la cifra cero que nos indica que queremos reiniciar la entrada.

El programa debe alterarse del modo siguiente:

```
500 FOR i = 1 TO 4
510 INPUT "Elemento "+STR$(i)+":";a(i)
```

```
520 IF 0<a(i) AND a(i)<7 THEN GO TO 560
530 IF a(i) = 0 THEN GO TO 500
540 PRINT "Cifra incorrecta"
550 GO TO 510
560 NEXT i
```

E...	1..2..3..4....	N..B
1	3 5 4 2	1 1
2	1 6 4 2	1 2
3	3 5 1 6	1 2
4	5 2 6 1	1 2
5	2 6 1 3	1 3
6	3 1 2 6	2 2
7	3 6 2 1	4 0

Ha acertado

Figura 1 Modelo de sistema de información

10.9.3.2 El sistema de información

El sistema de información es el que precisa B para un buen control del juego. Dado que en el ZX-SPECTRUM la entrada de datos se realiza en la parte baja de la pantalla podemos utilizar la parte alta para mantener la información restante.

Se utiliza una línea de pantalla para registrar el resultado de un ensayo. Consta de varias columnas en la que se registra la combinación de cifras entrada y el número de marcadores negros y blancos obtenidos.

Por ejemplo, la tabla de la figura 1 es un esquema del sistema de información.

Desde el punto de vista de programación nos obliga a:

1. Escribir la cabecera de pantalla antes de iniciar las entradas de datos.

2. Tener una variable que nos controle qué ensayo está realizando el jugador B.

3. Visualizar en la línea correcta la combinación entrada por el jugador B.

4. Escribir el resultado de los marcadores también en la línea correcta. De momento consideremos otra vez la parte de la entrada de datos. Para sincronizarse con el sistema de información debemos diseñar una fase previa de colocación de la tabla y el número de ensayo.

El programa quedaría entonces como:

```
500 CLS :REM Inicializacion pantalla
510 PRINT "E...1..2..3..4....N..B"
520 PRINT "------------------------"
530 REM Inicializacion variables
540 LET e=1 : LET p=3
600 REM Inicio del ensayo
610 PRINT AT p,0;"                            "
620 PRINT AT p,0;e
700 REM Entrada de datos
710 FOR i = 1 TO 4
720 INPUT a(i):PRINT AT 20,0;"                    "
730 IF a(i) = 0 THEN GO TO 600
740 IF 0<a(i) AND a(i)<7 THEN GO TO 770
750 PRINT AT 20,0;"Cifra Incorrecta"
760 GO TO 720
770 PRINT AT p,3*i+1;a(i)
780 NEXT i
```

La línea 500 nos limpia la pantalla, esto no sería necesario la primera vez pues el ZX-SPECTRUM lo hace automáticamente; sin embargo, cuando encadenamos un juego con el siguiente será necesario limpiar el juego anterior.

Las líneas 510 y 520 colocan la cabecera del sistema de información.

Las líneas 530 y 540 inicializan las variables *e* y *p*. La variable *e* es la que nos sirve para contar los ensayos y la variable *p* para saber en qué línea hay que escribir en el sistema de información. Se coloca la *e* inicialmente a 1 y la *p* a 3 para que se inicie en la cuarta línea, tenga en cuenta que el ZX-SPECTRUM inicia la pantalla en la línea 0. Se deja, por lo tanto, una línea en blanco entre la cabecera y el primer ensayo.

Las líneas 600, 610 y 620 son las que inician el ensayo imprimiendo en el sistema de información el ensayo que estamos realizando. Sorprende un poco la utilidad de la línea 610.

Simplemente escribe blancos sobre la línea de ensayo, esto se hace porque como ya hemos dicho cuando se teclea un cero en la entrada de datos queremos volver a iniciar la combinación. Entonces hay que dejar esta línea limpia de los datos anteriores.

De la línea 700 hasta la 780 es propiamente la entrada de datos que hemos diseñado ya. Sólo son necesarios los comentarios siguientes:

1. En el INPUT no se coloca la indicación del elemento pues el sistema de información nos indica claramente qué elementos estamos entrando.

2. El mensaje de «Cifra incorrecta» se visualiza en la línea 20 de la pantalla para no molestar en el sistema de información. Para que el mensaje no quede siempre colocado es necesario después del INPUT escribir blancos en esta línea. De esta manera cuando detectamos un error volvemos al INPUT, una vez hemos entrado el nuevo dato borramos la línea de pantalla nº 20 que contiene el mensaje de error y reemprendemos otra vez el análisis de la nueva cifra entrada.

3. La línea 770 nos escribe la cifra correcta entrada en su lugar en la tabla de información. Para ello se utiliza la fórmula 3*i+1, que calcula la columna en que hay que poner la cifra en función de la variable de control del bucle.

4. Cuando la cifra entrada es un cero, entonces el sistema vuelve a la línea 600 como si entráramos de nuevo en el ensayo.

Ya estamos en condiciones de hacer una prueba del programa

Recuerde que la instrucción 10, 100 y 110 están definidas en el apartado anterior.

Teclee el RUN y observará cómo funciona el mecanismo diseñado.

Es necesario que pruebe la entrada de cifras incorrectas y utilice el cero como inicio del ensayo.

10.9.3.3 El cálculo de los marcadores

Para el cálculo de los marcadores es necesario disponer de dos variables numéricas que cuenten los aciertos de un tipo u otro. Las denominaremos *n* para el recuento de los marcadores negros y *b* para el de los blan-

cos. Es decir *n* para los aciertos en color (cifra en nuestro caso) y posición y *b* para los aciertos en el color sólo.

Una vez colocadas las variables de los marcadores a cero hay que iniciar un repaso de la configuración oculta para ver el grado de coincidencia.

Esto se organiza en dos bucles anidados. El más externo para elegir cada uno de los elementos de la combinación oculta. El más interno compara el elemento de la combinación oculta seleccionado con cada uno de los elementos de la combinación de ensayo.

En el momento que se encuentran dos elementos iguales debe preguntarse si las posiciones de los elementos en el interior de sus respectivas combinaciones son coincidentes.

En el caso de que lo sean se incrementa la variable asociada al marcador negro, en caso contrario, la posición no es coincidente, se incrementa la variable del marcador blanco.

El programa que realiza esta comparación es el siguiente:

```
800 REM Calculo de marcadores
810 LET b = 0 : LET n = 0
820 FOR i=1 TO 4
830 FOR j = 1 TO 4
840 IF r(i) <> a(j) THEN GO TO 890
850 IF i <> j THEN GO TO 880
860 LET n= n+1
870 GO TO 890
880 LET b = b +1
890 NEXT j
900 NEXT i
910 PRINT AT p,18;n;"    ";b
```

La línea 810 inicializa las variables *b* y *n*.

La línea 820 abre el bucle externo que se cierra en 900. La variable *i* nos indica qué elemento de la combinación oculta estamos comparando.

La línea 830 abre el bucle interno que se cierra en 890. La variable *j* nos indica el elemento de la combinación de ensayo que estamos comparando.

La línea 840 compara la cifra de ambas combinaciones. Si son distintas envía el control del programa al final del bucle más interno.

En el caso de que las cifras sean iguales se alcanza la instrucción 850 que pregunta si los valores de *i*, posición en la combinación oculta, y *j*, posición en la combinación de ensayo son distintas, si la respuesta es afirmativa se incrementa la variable *b* en uno, se han detectado cifras iguales en las dos combinaciones pero que no están en la misma posición.

Si las posiciones *i* y *j* son iguales además de tener cifras iguales tenemos posiciones coincidentes hay que incrementar el marcador negro.

Una vez hemos hecho el repaso completo de todas las cifras de la combinación oculta con la combinación de ensayo alcanzamos la línea 910 en donde se escribe el resultado en el lugar correcto de la pantalla, indicado por la fila *p* y por la columna 18.

10.9.3.4 Encadenamiento de los ensayos

Tenemos ya completa la entrada de una configuración de ensayo y el diagnóstico respecto a la configuración oculta.

Es necesario para que el juego pueda funcionar encadenar un ensayo al siguiente. Los ensayos se inician en la línea 600 tal como hemos señalado.

Las variables de control son *e* y *p*, la primera nos indica el número de ensayo y la segunda el lugar de la línea de información.

Una vez finalizado el ensayo lo primero que hay que hacer es incrementar estas variables.

A continuación hay que decidir si el juego ha finalizado. Esta finalización puede ocurrir ya sea porque se ha sobrepasado el límite ya sea porque se ha acertado la combinación oculta.

La primera condición corresponde al caso de que el jugador B pierde y la segunda la que el jugador B gana.

Las instrucciones siguientes realizan estas funciones:

```
1000 LET e = e+1 : LET p=p+1
1010 IF e<10 AND n<4 THEN GO TO 600
1100 REM Finalizacion del juego
1110 IF n=4 THEN GO TO 1200
1120 PRINT AT p,1;"Lo siento. Ha perdido"
1130 GO TO 2000
1200 PRINT AT p,1;"Muy bien. Ha ganado"
```

La línea 1000 incrementa las variables de control.

La línea 1010 se hace la pregunta de si no se ha alcanzado el final de la partida. Se pregunta, por un lado si $e<10$ que corresponde a un tope de ensayos fijados de antemano para acertar la combinación. Por otro, si el marcador de negros es menor que cuatro, con *n* igual a cuatro se ha acertado la combinación. Se encadenan las dos preguntas y si ambas son ciertas se autoriza un nuevo ensayo.

En caso de que no se sigan los ensayos las instrucciones 1100 hasta la 1200 dan el diagnóstico del final de la partida. Este diagnóstico se basa en el resultado de si *n* vale cuatro que es el caso de que se ha ganado.

El número 10, es decir, el límite de ensayos permitidos son los que colocan la dificultad del juego a un nivel determinado. Por lo tanto, variando este número se puede hacer el juego más fácil o difícil, probablemente el valor que da una dificultad razonable al juego es de 6 ó 7.

10.9.3.5 Obtención de la combinación oculta

El mecanismo de selección de la combinación oculta puede ser automático o manual.

El procedimiento automático consiste en utilizar una función que dispone el ordenador para generar números al azar. Este mecanismo aún no se ha estudiado en esta lección, se verá en una lección posterior.

Por lo tanto, optamos en este caso para el mecanismo de selección manual. En este tipo de generación la entrada de las cifras se hace mediante el teclado. En el caso de generación automática bastará sustituir esta llamada al teclado por la llamada a otra función. Volveremos a este mecanismo en el momento oportuno.

La idea de entrar la combinación oculta es el mismo que hemos discutido para la entrada de la combinación de ensayo.

En este caso, sin embargo, no deberemos utilizar toda la parte de marcadores ni tampoco encadenar los ensayos sucesivos. Será necesario, por otra parte, borrar la pantalla con la combinación oculta para que efectivamente se mantenga oculta al jugador B durante los ensayos. También será necesario comprobar que no se entren cifras repetidas pues las reglas del juego impiden que haya dos cifras iguales en la configuración oculta.

La líneas de programación son las siguientes:

```
100 REM Seleccion Configuracion
110 CLS : PRINT "Configuracion secreta"
120 PRINT "..1..2..3..4.."
130 PRINT "--------------"
140 PRINT "              "
200 FOR i = 1 TO 4
210 INPUT r(i): PRINT AT 20,0;"                               "
220 IF r(i) = 0 THEN GO TO 100
230 IF 0<r(i) AND r(i)<7 THEN GO TO 300
240 PRINT AT 20,0;"Cifra incorrecta"
250 GO TO 210
260 PRINT AT 20,0;"Cifra repetida"
270 GO TO 210
300 FOR j = 1 TO i-1
310 IF r(i) = r(j) THEN GO TO 260
320 NEXT j
330 PRINT AT 3,3*i-1;r(i)
400 NEXT i
```

De la línea 100 hasta la 140 sirven para borrar la pantalla y preparar la cabecera para indicar la posición de los elementos.

En la línea 200 se inicia el bucle para cada una de las posiciones de la combinación oculta. Este bucle finaliza en la línea 400.

La línea 210 es el INPUT que nos pide la cifra y el borrado de la línea de error tal como explicábamos en el apartado de la entrada de las combinaciones.

La línea 220 analiza si la cifra es cero, en este caso envía el control del programa a reiniciar la entrada de la configuración oculta.

La línea 230 comprueba si la cifra está comprendida entre los márgenes adecuados en este caso envía el control a la línea 300.

Si la cifra no es correcta continúa por la línea 240 que da el mensaje de error y se recicla a la línea 210, nueva entrada de la cifra.

Las líneas 260 y 270 es la escritura del mensaje de error de cifra repetida y posterior reciclaje a una nueva entrada. A esta línea accederemos desde más abajo en el bucle.

En la línea 300 se inicia el mecanismo de comprobación de si las cifras están repetidas.

Para ello se inicia el bucle cuya variable de control es j. La variable de control se inicia con el valor de 1 y con un valor final de i–1 que es el correspondiente a la posición anterior a la que estamos entrando.

Evidentemente cuando entramos la primera cifra este valor final vale cero y por lo tanto el bucle no se recorre ninguna vez. En la primera cifra no se ha de hacer ninguna comprobación.

El interior del bucle consiste en una decisión de tal manera que si la cifra entrada es igual a una de las cifras anteriores el programa continúa por la línea 260 que corresponde al error de cifra repetida.

Una vez se ha alcanzado el final del bucle obtenemos una cifra correcta. Se imprime a continuación la cifra en el lugar correspondiente a la pantalla (línea 330) mediante una función que depende de la variable de control del bucle más externo que es la posición que estamos entrando.

La línea 400 nos cierra el bucle iniciado en la línea 200. A partir de este momento interviene ya el jugador B.

10.9.3.6 Encadenamiento de partidas

El último paso consiste en que cuando finaliza el juego el programa nos pregunta si deseamos jugar otra partida y en este caso se recicla al inicio de una nueva configuración oculta.

No se debe repetir la instrucción que contiene la dimensión pues no debe ejecutarse dos veces.

El resto del programa es:

```
2000 INPUT "Quiere jugar otra partida(s/n)";a$
2010 IF a$<> "n" THEN GO TO 100
```

10.9.4 Observaciones finales

Indicar, finalmente, que se trata de un juego divertido y que el programa le permitirá pasar ratos agradables.

Desde el punto de vista de programación es un programa interesante pues aparecen operaciones relativamente complicadas con las tablas.

Observe que las tablas y los FOR/NEXT son compañeros inseparables.

Estudie con detalle el mecanismo de comprobar la repetición en la obtención de la configuración oculta y el mecanismo de obtención de los marcadores son nuevos procedimientos básicos para la manipulación de tablas.

Finalmente guarde el programa en un cassette con el nombre de «mind».

El listado completo del programa se da a continuación.

```
  10 DIM r(4) : DIM a(4)
 100 REM Seleccion Configuracion
 110 CLS : PRINT "Configuracion secreta"
 120 PRINT "..1..2..3..4.."
 130 PRINT "--------------"
 140 PRINT "              "
 200 FOR i = 1 TO 4
 210 INPUT r(i): PRINT AT 20,0;"                                "
 220 IF r(i) = 0 THEN GO TO 100
 230 IF 0<r(i) AND r(i)<7 THEN GO TO 300
 240 PRINT AT 20,0;"Cifra incorrecta"
 250 GO TO 210
 260 PRINT AT 20,0;"Cifra repetida"
 270 GO TO 210
 300 FOR j = 1 TO i-1
 310 IF r(i) = r(j) THEN GO TO 260
 320 NEXT j
 330 PRINT AT 3,3*i-1;r(i)
 400 NEXT i
 500 CLS :REM Inicializacion pantalla
 510 PRINT "E...1..2..3..4....N..B"
 520 PRINT "----------------------"
 530 REM Inicializacion variables
 540 LET e=1 : LET p=3
 600 REM Inicio del ensayo
 610 PRINT AT p,0;"                              "
 620 PRINT AT p,0;e
 700 REM Entrada de datos
 710 FOR i = 1 TO 4
 720 INPUT a(i):PRINT AT 20,0;"                                "
 730 IF a(i) = 0 THEN GO TO 600
 740 IF 0<a(i) AND a(i)<7 THEN GO TO 770
 750 PRINT AT 20,0;"Cifra Incorrecta"
 760 GO TO 720
 770 PRINT AT p,3*i+1;a(i)
 780 NEXT i
 800 REM Calculo de marcadores
 810 LET b = 0 : LET n = 0
 820 FOR i=1 TO 4
 830 FOR j = 1 TO 4
 840 IF r(i) <> a(j) THEN GO TO 890
 850 IF i <> j THEN GO TO 880
 860 LET n= n+1
 870 GO TO 890
 880 LET b = b +1
 890 NEXT j
 900 NEXT i
 910 PRINT AT p,18;n;"  ";b
1000 LET e = e+1 : LET p=p+1
```

```
1010 IF e<10 AND n<4 THEN GO TO 600
1100 REM Finalizacion del juego
1110 IF n=4 THEN GO TO 1200
1120 PRINT AT p,1;"Lo siento ha perdido"
1130 GO TO 2000
1200 PRINT AT p,1;"Muy bien. Ha ganado"
2000 INPUT "Quiere jugar otra partida (s/n)";a$
2010 IF a$<> "n" THEN GO TO 100
```

10.10 PRACTICA 2. TABLA DE CONVERSION DE MONEDAS

10.10.1 Objetivo

La práctica consiste en realizar un programa que dada una cantidad de dinero en ciertas unidades monetarias, calcule el valor en otras unidades monetarias.

Supongamos que trabajamos con las siguientes unidades monetarias: pesetas, dólares USA, libras esterlinas, marcos alemanes y yens japoneses.

El ordenador debe responder a la pregunta de ¿Cuánto son 2 dólares USA en valor del resto de unidades monetarias?

Es decir, debe calcular una lista con los valores siguientes:

313 Pesetas, 2 Dólares USA, 1.4 Libras Esterlinas, 5 Marcos Alemanes y 405 Yens Japoneses.

Naturalmente para conseguir este objetivo es necesario una tabla de conversión entre las distintas unidades monetarias.

Además se requiere del programa otra propiedad importante, que es que la tabla de unidades monetarias sea totalmente libre, con lo que el usuario del programa puede elegir las monedas que más le convengan.

10.10.2 Escritura del programa

El programa se divide en tres partes:

1. Entrada de la tabla de conversiones.
2. Entrada de las preguntas respecto a las cantidades.
3. Cálculo y presentación de los resultados.

10.10.2.1 Tabla de conversiones

Para realizar esta tabla es necesario definir una unidad monetaria básica. El cambio de las demás monedas se introduce respecto a la unidad básica.

Utilizamos dos conjuntos dimensionados de una dimensión. El primero numérico, que sirve para expresar los valores de cambio en función de la unidad monetaria básica y el segundo alfabético para almacenar el nombre de la moneda.

Como la tabla es del tamaño elegido por el usuario es necesario preguntar primero cuántas monedas se van a utilizar en cada ejecución del programa.

Por otra parte, la primera unidad monetaria es la unidad monetaria básica. En ella sólo es necesario entrar el nombre ya que el valor de la conversión es 1.

Finalmente hay que entrar el nombre y el valor del cambio para el resto de monedas a considerar.

Para tomar un ejemplo, utilizamos las siguientes unidades monetarias y los valores de conversión siguientes:

Pesetas 1

Dólares 156 (USA)

Libras 223 (Esterlina)

Marcos 63 (Alemán)

Yens 0.77 (Japonés)

En la tabla se han simplificado adrede los nombres para evitar manipular nombres demasiado complicados. Entre paréntesis se han colocado los calificativos necesarios para especificar correctamente la moneda.

Se considera que el nombre de las monedas se puede expresar con 10 caracteres.

La parte del programa que realiza la entrada de esta tabla es el siguiente:

```
100 REM Entrada de la Tabla.
110 INPUT "Cuantas monedas va entrar:";n
120 DIM c(n):DIM m$(n,10)
200 REM Entrada de la moneda base
210 INPUT "Nombre de la moneda base:";m$(1)
220 LET c(1) = 1.
300 REM Entrada del resto de la tabla
310 FOR i = 2 TO n
320 INPUT "Nombre de las otras monedas:";m$(i)
330 INPUT "Valor en la moneda base:",c(i)
340 NEXT i
```

Las líneas 110 y 120 nos permiten dimensionar los conjuntos de manera adecuada.

La línea 110 nos pide el valor y la 120 dimensiona el conjunto *c* que será el conjunto numérico que contiene el valor de las monedas. También

dimensiona el conjunto *m$* que contiene el nombre de las distintas monedas con 10 caracteres como máximo de longitud de la cadena.

Las instrucciones 210 y 220 asignan el nombre de la moneda base y el valor de conversión de la moneda base que obligatoriamente es 1. La asignación del nombre se hace mediante una instrucción INPUT.

Las instrucciones 310, 320 y 330 forman un bucle que va pidiendo el nombre y el valor de las distintas monedas a considerar. El bucle va desde un valor inicial de 2 hasta el valor *n*. Recuerde que el primer elemento de la tabla es el de la moneda base entrado anteriormente.

10.10.2.2 Mecanismo de pregunta

La siguiente parte del programa consiste en la pregunta de las conversiones que se desean realizar.

Esta pregunta consiste en pedir la cantidad en unidades monetarias de una moneda determinada.

Para finalizar el programa correctamente es necesario definir una condición para finalizar. Esta condición es que la cantidad entrada sea negativa o nula. En general, no tiene sentido querer convertir una cantidad de moneda negativa o nula.

Una vez entrada la moneda hay que comprobar que la moneda está en la tabla que tenemos en memoria. Se localiza en la tabla por nombre. Si el resultado es encontrado, se prosigue el programa; en caso contrario, se da un mensaje de error y se vuelve a pedir la moneda.

Hay que recordar que, para buscar en la tabla por nombre, el ZX-SPECTRUM requiere que la petición de nombre sea de la misma longitud que el definido en la dimensión. Por ello, se rellena con blancos el nombre de la moneda que entramos.

El programa es:

```
400 REM Peticion y busqueda
410 INPUT "Cantidad que desea cambiar:";q
420 IF q<=0 THEN GO TO 9999
500 INPUT "Moneda que desea cambiar:";a$
510 IF LEN a$ < LEN m$(1) THEN LET a$=a$+" " : GO TO 510
600 REM Busqueda por Nombre
610 FOR i = 1 TO n
620 if a$ = m$(i) THEN GO TO 700
630 NEXT i
640 CLS:PRINT "No encuentro la moneda"
650 GO TO 500
```

Las línea 410 pide la cantidad. La línea 420 analiza si la cantidad es negativa o nula, en caso afirmativo finaliza el programa. En caso negativo continúa el programa.

La línea 500 pide el nombre de la moneda y la línea 510 alarga este nombre hasta la misma longitud de los nombres definidos en la tabla, se utiliza el primer elemento para evitar que un cambio en la dimensión nos haga modificar también esta instrucción.

La línea 610 inicia el bucle de búsqueda que se cierra en la línea 630. La línea 620 es una pregunta que comprueba si el nombre entrado es igual

al elemento de la tabla que está en la posición *i*. En caso afirmativo se envía el programa a la línea 700 para realizar el cálculo.

Si se finaliza el bucle significa que no hay ningún elemento en la tabla que coincida con el nombre entrado.

La línea 640 nos informa del error y la línea 650 nos envía a preguntar por un nuevo nombre.

10.10.2.3 Escritura de los resultados

Una vez alcanzado este punto del programa, sólo es necesario calcular las conversiones y escribirlas en pantalla.

El cálculo de las conversiones se hace transformando la cantidad entrada a unidades monetarias bases. El mecanismo es multiplicar la cantidad entrada *q* por el valor de cambio del nombre localizado en la tabla que será *c(i)*.

Se inicia entonces un bucle para todas las monedas. En cada una de ellas se calcula el valor mediante la división de la cantidad expresada en unidades monetarias de la moneda base por el valor de cambio de una unidad de la moneda base, que está contenida en *c(j)* (*j* es la variable de control de este bucle de escritura. Observe que podría llamarse también *i* puesto que ya hemos incorporado en la cantidad *q* el valor de la moneda pedido).

Finalmente, se escribe en la pantalla el valor obtenido y el nombre.

Cuando se sale de bucle se envía el programa a una nueva pregunta.

La escritura de esta parte del programa es:

```
700 CLS : LET q = q * c(i)
710 FOR j = 1 to n
720 PRINT q/c(j);" ";m$(j)
730 NEXT j
740 GO TO 400
```

La línea 700 borra la pantalla y calcula el nuevo valor de *q* en función de las unidades monetarias básicas. Se utiliza para esto el valor de cambio c(i), el valor de i viene definido por la variable de control del bucle de búsqueda del apartado anterior. Necesariamente para alcanzar la línea 700 hay que salir del interior del bucle de búsqueda por nombre.

La línea 710 inicia un bucle para todas las monedas que se cierra en 730.

La línea 720 imprime el valor q en las unidades monetarias correspondientes a la moneda en la posición *j* de la tabla y a continuación escribe el número.

Finalmente el programa se envia a la línea 400 para que se realice otra pregunta.

El listado completo del programa se da a continuación:

```
100 REM Entrada de la Tabla.
110 INPUT "Cuantas monedas va a entrar:";n
120 DIM c(n):DIM m$(n,10)
200 REM Entrada de la moneda base
210 INPUT "Nombre de la moneda base:";m$(1)
220 LET c(1) = 1.
300 REM Entrada del resto de la tabla
310 FOR i = 2 TO n
320 INPUT "Nombre de las otras monedas:";m$(i)
330 INPUT "Valor en la moneda base:",c(i)
340 NEXT i
400 REM Peticion y busqueda
410 INPUT "Cantidad que desea cambiar:";q
420 IF Q<=0 THEN GO TO 9999
500 INPUT "Moneda que desea cambiar:";a$
510 IF LEN a$ < LEN m$(1) THEN LET a$=a$+" " : GO TO 510
600 REM Busqueda por Nombre
610 FOR i = 1 TO n
620 if a$ = m$(i) THEN GO TO 700
630 NEXT i
640 CLS:PRINT "No encuentro la moneda"
650 GO TO 500
700 CLS : LET q = q * c(i)
710 FOR j = 1 to n
720 PRINT q/c(j);" ";m$(j)
730 NEXT j
740 GO TO 400
```

10.10.3 Pruebas

Para finalizar la práctica realice las pruebas siguientes:

1. Entre la tabla que se ha mencionado más arriba en la que intervienen pesetas, dólares, libras, marcos y yens. Es aconsejable que entre los elementos de la tabla con todas las letras minúsculas y en plural, tal como hemos escrito en este párrafo. La presentación de los datos finales es más correcta en plural y más difícil equivocarse si se hace todo en minúsculas.

2. Cuando le pida la cantidad, dé 2 de la moneda dólares. La tabla que debe obtener es:

312 pesetas
2 dólares
1.3991031 libras
4.952381 marcos
405.19481 yens

3. Introduzca ahora 5 de cantidad y al pedir el nombre entre liras, moneda que no está en la tabla. El programa debe darle el error que no ha encontrado la moneda y preguntarle otra moneda. Entre ahora libras.
El resultado debe ser:

1115 pesetas
7.1474359 dólares
5 libras
17.698413 marcos
1448.0519 yens

4. Guarde el programa en un cassette con el nombre «cambio».

Capítulo 11

ESQUEMA DE CONTENIDO

Cómo almacenar datos

Instrucción DATA

Instrucción READ

Instrucción RESTORE

Aplicaciones

El mes en letras

El día en letras

La agenda

Funciones definidas por el programador

Práctica 1. Realización de la factura
- Definición del programa
- Diseño general
  - Mecanismos
  - Estructura de datos
- Escritura del programa
  - Carga de la lista de precios
  - Entrada de la fecha y del impuesto
  - Fichero de clientes
  - Tabla de líneas
  - La escritura de la factura
- Consideraciones finales

Práctica 2. Dibujar una función
- Consideraciones generales
- Escritura del programa

11.1 COMO ALMACENAR DATOS

En el programa utilizado para realizar la lista de notas del capítulo estándar debemos cambiar la forma de indicar las dimensiones. En el ZX-Spectrum, los conjuntos dimensionados de tipo textual necesitan una especificación de las dimensiones distinta de la estándar, puesto que en ella hay que indicar cuántas letras contiene cada elemento. Además es necesaria para cada conjunto dimensionado una instrucción DIM.

Por lo tanto, la línea 50 se debe cambiar por

```
50 DIM N$(NA,9) : DIM T(NA)
```

Se pueden poner expresiones textuales para el mensaje del INPUT

La segunda dimensión de N$ indica el número de letras de cada elemento en este caso 9. Se ha escogido este valor pues corresponde al nombre de alumno más largo. El conjunto numérico no sufre variación.

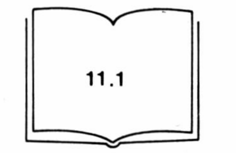
Por otra parte, en la entrada de las notas se utiliza primero un PRINT (línea 150) y luego un INPUT (línea 160), porque el BASIC estándar no permite que en el INPUT se utilice una expresión textual y solamente permite una cadena de caracteres constante.

Sin embargo, el ZX-Spectrum permite que en el INPUT se calculen expresiones, por lo tanto, es mejor sustituir las líneas 150 y 160 por la única 150 siguiente:

```
150 INPUT "Entre la nota de "+N$(I)+":";T(I)
160             <-------------- BORRAR
```

En la instrucción INPUT de la línea 150 se encadenan las cadenas de caracteres mediante el operador de concatenación.

El resto del programa queda idéntico y funciona tal como está escrito.



11.2 INSTRUCCION DATA

La palabra DATA se encuentra escrita sobre la tecla D y se accede a ella mediante el EXTEND MODE. A continuación de esta instrucción se coloca una lista de datos separados por comas.

Siempre hay que colocar los datos textuales entre comillas

En la mayoría de variantes del BASIC es opcional la colocación de comillas en los datos textuales excepto en aquellos que contienen la coma. En el ZX-Spectrum esto no está permitido. Siempre que debamos colocar un texto en un DATA deberemos utilizar las comillas. Por ejemplo:

```
DATA "CASA"
DATA "Fernandez","Garcia"
DATA "c/ Virgen de Guadalupe, 23 3-4","Avda. Luz, 234 Bajos"
```

Una coma que forme parte de un texto no origina confusión ya que está englobada entre comillas y se distingue perfectamente de la coma de separación entre datos. Observe en el último ejemplo que las comas después de Guadalupe y Luz son parte del texto porque están encerradas entre comillas.

Se pueden colocar expresiones que pueden incluir variables del programa

Como contrapartida de esta limitación, el ZX-Spectrum permite colocar como dato una expresión que evaluará en el momento de leer el dato, el modo de leer lo veremos en el apartado siguiente.

Es decir, son válidos como datos detrás de un DATA expresiones válidas en BASIC. Cuando en una expresión aparece un nombre se considera que se trata de una variable que está definida. Si no lo está se comete el error de variable no definida.

Las siguientes líneas son muestras de datos en forma de expresión:

```
DATA 2*4,3+6,"Juan"+"Pedro"
DATA 2*a,3+y,a$+"Pedro"
```

en la primera de ellas se utilizan dos expresiones numéricas y una alfanumérica. En la segunda se utilizan las mismas expresiones que en la primera, pero se han sustituido ciertas constantes por variables, así el 4 se ha sustituido por *a*, el 6 se ha sustituido por *y* y «Juan» se ha sustituido por *a$*. Si en el programa no están definidas todas estas variables cuando se utilicen estos datos se detecta error.

La línea siguiente presenta un caso engañoso

```
DATA "Juan","Pedro",Juan,Pedro
```

en efecto, los dos primeros datos son textuales y equivalen a las cadenas Juan y Pedro respectivamente. Los dos últimos datos también son las cadenas Juan y Pedro pero sin comillas. Ya sabemos que no pueden ser datos textuales, es la primera norma que hemos explicado en este apartado. ¿Qué son entonces? pues bien el ZX-Spectrum las considera variables numéricas que cuando vaya a utilizar a estos datos las quiere definidas.

Hemos calificado este caso de engañoso porque muchas veces uno quiere preparar un DATA con datos textuales y lo hace sin comillas como se hace en muchos BASIC. El ZX-Spectrum no protesta hasta el momento de la ejecución en donde nos repetirá el error variable no definida.

11.3 LA INSTRUCCION READ

La palabra READ se encuentra escrita sobre la tecla A y se accede a ella mediante el EXTEND MODE.

La manera de escribir esta instrucción es idéntica a la del BASIC estándar.

El mecanismo de comportamiento también es muy parecido al del BASIC estándar. Hay que hacer la salvedad de que es capaz de leer expresiones (como se ha indicado en el apartado anterior) contenidas en las instrucciones DATA, en este sentido se comporta exactamente igual que una instrucción INPUT del ZX-Spectrum con la diferencia de que en lugar de leer del teclado lee del propio programa.

Aunque el ZX-Spectrum ofrece la posibilidad de utilizar expresiones, no es recomendable hacerlo para que el programa sea compatible con otros BASIC; ciertamente ésta es una propiedad que sólo tiene el ZX-Spectrum.

Errores que se producen en la instrucción READ

Los errores que se producen en esta instrucción son de dos tipos. El primero que nos informa de que se *intenta leer más allá del último DATA*. Se trata de

E Out of Data, 20:1

y el segundo es la serie de errores de conversión, tales como *Sin Sentido en BASIC*, si queremos leer un número y encontramos un texto o viceversa, *variable no definidas* si utilizamos instrucciones DATA con expresiones, etc.

En el programa en el que se utilizan datos mezclados, tenga en cuenta que el Spectrum no añade el espacio en blanco después del dato numérico. Déle usted, por tanto un espacio antes y después de las comillas.

En el programa de las notas de los alumnos correspondiente al apartado de la instrucción READ de la lección estándar son válidas las observaciones siguientes:

1. La línea 50 debe ser

```
50 DIM N$(NA,9) : DIM T(NA)
```

2. Las líneas 150 y 160 se deben sustituir por la única 150 siguiente:

```
150 INPUT "Entre la nota de "+N$(I)+":";T(I)
160            <------------- BORRAR
```

11.4 LA INSTRUCCION RESTORE

La palabra RESTORE se encuentra escrita sobre la tecla S y se accede a ella mediante el EXTEND MODE.

Rebobinados parciales

En el ZX-Spectrum se permite realizar «rebobinados parciales», es decir que, en lugar de empezar otra vez desde la primera línea DATA, se continúa a partir de una línea determinada.

Para efectuar esta operación de rebobinado parcial es necesario colocar detrás de la instrucción RESTORE el número de línea que será el inicio.

Por ejemplo,

```
RESTORE 100
```

indica que se rebobina hasta el primer dato de la primera instrucción DATA que está en la línea 100 o siguientes.

Naturalmente después de la instrucción anterior, una instrucción READ lee el primer dato de la primera instrucción DATA que está en la línea 100 o siguientes.



11.5 APLICACIONES

11.5.1 El mes en letras

En el programa utilizado para transformar la fecha en letras introduciremos algunas variaciones:

— En primer lugar la línea 30 debe quedar como

```
30 DIM N$(12,10)
```

el ZX-Spectrum requiere que las variables textuales dimensionadas lleven el número máximo de caracteres que va a contener el texto. El segundo número (10) establece que cada elmento de N$ contiene 10 letras. La elección de este valor se debe a que corresponde al número de letras de «Septiembre» que es el nombre del mes más largo.

— En segundo lugar es necesario cambiar las funciones LETF$, MID$ y RIGHT$. En el ZX-Spectrum escribiremos en su lugar

```
110 LET D$=F$(1 TO 2)
120 LET M$=F$(4 TO 5)
130 LET A$=F$(7 TO 8)
```

Las instrucciones DATA se consideran como comentarios en el momento de la ejecución

— Finalmente, la instrucción END de la línea 220 la cambiaremos por un STOP o por un GO TO a un número de línea mayor que los existentes. No obstante, las instrucciones DATA no interfieren en la ejecución. El BASIC sencillamente las ignora y pasa a la siguiente línea. Por lo tanto, la línea 220 puede suprimirse si se desea.

Las instrucciones DATA actúan de almacén y sólo las detectan las instrucciones READ y RESTORE por lo demás son como comentarios al programa.

11.5.2 El día en letras

En la ampliación del programa que permite la conversión de las cifras del día a un texto, realizaremos las siguientes modificaciones:

— La línea 30 debe cambiarse a

```
30 DIM N$(12,10) : DIM P$(22,10)
```

La causa de esta modificación es la necesidad de especificar la longitud máxima de la cadena de caracteres cuando el conjunto dimensionado es de variables textuales.

— La línea 166 se cambia por

```
166 LET R=VAL(D$(2 TO 2))
```

11.5.3 La agenda

En el programa de la agenda debemos realizar los cambios siguientes:

— Las dimensiones de la tabla A$ se definen así:

```
30 DIM A$(50,3,22) : LET I=0
```

El valor de 22 corresponde a la longitud del nombre mayor.

Hay que rellenar las entradas con blancos

— Los elementos que contengan menos letras se rellenarán en la tabla con blancos. Esta propiedad introduce una complicación en la comparación de la línea 340.

En el capítulo anterior ya hemos visto que es necesario añadir a la entrada tantos blancos como sean necesarios para completar la longitud adecuada. Las instrucciones a añadir son:

```
295 IF LEN(D$)<LEN(A$(1,1)) THEN LET D$=D$+" " : GO TO 295
```

se coloca el elemento A$ (1,1) para hacerlo inmune a un retoque en la longitud de la instrucción DIM.

— También es posible que la función INKEY$ nos cause problemas debido a la memorización de las teclas. Cambiaremos la línea 400 por

```
400 PAUSE 0: IF INKEY$="" THEN GOTO 400
```

— Finalmente, después que usted haya escrito el programa completo de la lección estándar observará que cuando escribe, por ejemplo *veintiuno* en su ordenador le saldrá escrito *veinti   uno*. Es decir, que ha dejado una serie de espacios. La razón, como ya habrá podido suponer es que hemos reservado 10 espacios para la variable P$.

A continuación le presentamos el programa del capítulo estándar, al que hemos añadido las instrucciones correspondientes para que anule estos espacios en blanco. Son concretamente las líneas 140 a 170. Además, como no dispone de números intermedios para incluir las nuevas líneas hemos renumerado de la línea 160 en adelante. De todas maneras no introduzca esta versión del programa en el ordenador hasta que haya escrito el correspondiente del capítulo estándar.

El programa corregido es el siguiente:

```
 140 REM Traduccion del mes
 150    LET M = VAL(M$)
 160    LET M$ = N$(M)
 170 REM Traduccion del dia
 180   LET V = VAL(D$)
 190   IF V>20 THEN GOTO 220
 200        LET D$ = P$(V)
 210        GOTO 340
 220   LET R = VAL(RIGHT$(D$,1))
 230   IF V>29 THEN GOTO 260
 240        FOR I = 10 TO 1 STEP -1
 250             IF P$(21)(I TO I) <> " " THEN GO TO 270
 260        NEXT I
 270        LET D$ = P$(21)(1 TO I+1)+P$(R)
 280        GOTO 340
 290   FOR I = 10 TO 1 STEP -1
 300       IF P$(22)(I TO I) <> " " THEN GO TO 320
 310    NEXT I
 320   LET D$ = P$(22)(1 TO I)
 330   IF R>0 THEN LET D$ = D$+" y "+P$(R)
 340 REM Construccion del resultado
 350    LET R$ = D$+" de "+M$+" de 19"+A$
 360    PRINT R$
 370    INPUT "Desea continuar (S/N): ",R$
 380    IF R$="S" THEN GOTO 90
 390 END
1000 DATA "Enero", "Febrero", "Marzo"
1010 DATA "Abril", "Mayo", "Junio"
1020 DATA "Julio", "Agosto", "Septiembre"
1030 DATA "Octubre", "Noviembre", "Diciembre"
2000 DATA "Uno","Dos","Tres","Cuatro","Cinco","Seis","Siete"
2010 DATA "Ocho","Nueve","Diez","Once","Doce","Trece"
2020 DATA "Catorce","Quince","Dieciseis","Diecisiete"
2030 DATA "Dieciocho", "Diecinueve", "Veinte"
2040 DATA "Veinti","Treinta"
```

11.6 FUNCIONES DEFINIDAS POR EL PROGRAMADOR

Las funciones definidas por el programador se construyen en el ZX-Spectrum mediante las palabras reservadas

DEF FN y FN

situadas sobre las teclas de los números 1 y 2 y a las que se accede mediante el EXTEND MODE y después pulsando el SYMBOL SHIFT.

Como se trata de palabras clave del ZX-Spectrum después de FN, ya sea en la definición como en la utilización, queda siempre un espacio en blanco, en general, no es molestia alguna pues lo pone el sistema automáticamente.

En el ZX-Spectrum los nombres de las funciones sólo pueden tener una letra de longitud, en este sentido es parecido a los conjuntos dimensionados.

Las funciones deben tener el nombre y los argumentos de 1 letra

También los argumentos que se utilizan en la definición de las funciones deben tener una sola letra. Cuando utilizamos las funciones este problema no existe pues se admite como argumento cualquier expresión válida en BASIC.

La definición de la función (instrucción DEF) puede estar situada en cualquier lugar del programa, incluso al final. El BASIC se encarga de buscar la definición entre todas las líneas cuando la necesita.

En la utilización de las funciones debe utilizar siempre la palabra reservada FN que está sobre la tecla 2. Si teclea las letras F y luego la N el resultado no es llamar a una función.

Los errores que tiene el ZX-Spectrum relacionados con las funciones definidas por el programador son dos:

a) Función que no se ha definido.

Si pretendemos utilizar una función de la que no hay instrucción DEF FN se obtiene el error

P FN without DEF

que significa utilización de función sin definición.

b) Discrepancia en los parámetros.

Cuando los argumentos en la utilización de una función no coinciden con los de la definición se obtiene el error

Q Parameter error

El error se obtiene tanto si hay discrepancia en número como en tipo. Por ejemplo, si se requiere uno textual y se llama con uno numérico o viceversa.



11.7 PRACTICA 1. REALIZACION DE UNA FACTURA

En el primer capítulo de la Enciclopedia se proponia un modelo de factura, y en los capítulos 4 y 5 se desarrollaban programas primitivos para construir aquella factura.

En este capítulo vamos a volver al problema con cambios importantes respecto a los programas mencionados.

La idea más importante que se va a utilizar en el desarrollo de este programa es utilizar códigos para los clientes y para los artículos para que la entrada de datos sea lo más ligera posible.

Esta posibilidad de codificación es posible aquí porque conocemos la manera de almacenar datos fijos en el propio programa. En realidad se trata de un esquema muy parecido al que se utiliza con los ficheros de datos en discos magnéticos. Como no disponemos de estos discos utilizamos la propia memoria del ordenador para simularlo mediante las instrucciones DATA.

La similitud del mecanismo utilizado en esta práctica con el que se utiliza con discos magnéticos le dan un valor adicional que conviene que Ud. sepa para prestar más atención si cabe.

11.7.1 Definición del problema

Le recomendamos que tome el capítulo 1 del primer tomo y busque la figura 1 para tenerla próxima durante la realización de la práctica.

La figura 1 muestra la maqueta de la factura.

La disposición espacial es muy parecida a la utilizada en los capítulos 4 y 5 del primer y segundo tomos, respectivamente.

Hay que recordar que se escribe únicamente la parte de la factura escrita a mano en la figura del capítulo 1. Las partes impresas ya están en todos los papeles que utilicemos. Como no disponemos de impresora, utilizaremos la pantalla, como ya hemos hecho en las anteriores ocasiones.

El programa debe hacer una remesa de facturas de diversos clientes.

Se debe disponer de un fichero de clientes que permita acceder al cliente mediante un código numérico.

Se dispondrá en memoria de una lista de precios que permita una entrada de un código de producto y una cantidad. Mediante el código el programa debe averiguar la descripción del producto y el precio.

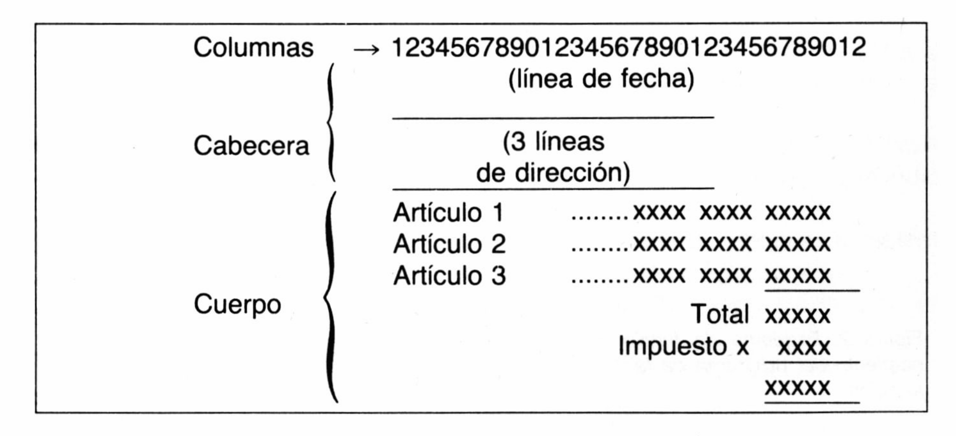


Figura 1. Maqueta de la distribución de los datos en la factura

11.7.2 Diseño general

11.7.2.1 Mecanismos

La figura 2 muestra el esquema general de funcionamiento del programa.

Como puede observar la primera parte consiste en cargar la lista de precios. La colocaremos entera en memoria para tener más facilidad de acceso.

Luego se pide la fecha y el valor del impuesto, son los mismos para todas las facturas.

A continuación se pregunta para qué cliente desea hacerse la factura (En el esquema se indica la pregunta con el símbolo ??).

Si la contestación es un marcador de final, usualmente se utiliza el código cero como marcador de final, el programa termina.

Si la contestación es el marcador de final va a buscar este cliente en el fichero de clientes.

El resultado de esta búsqueda puede ser encontrado y no encontrado. En el esquema estos dos posibles resultados se dibujan mediante líneas paralelas pues o bien se sigue una o la otra, pero no ambas a la vez. (También se utiliza el símbolo ?? para indicar este doble resultado).

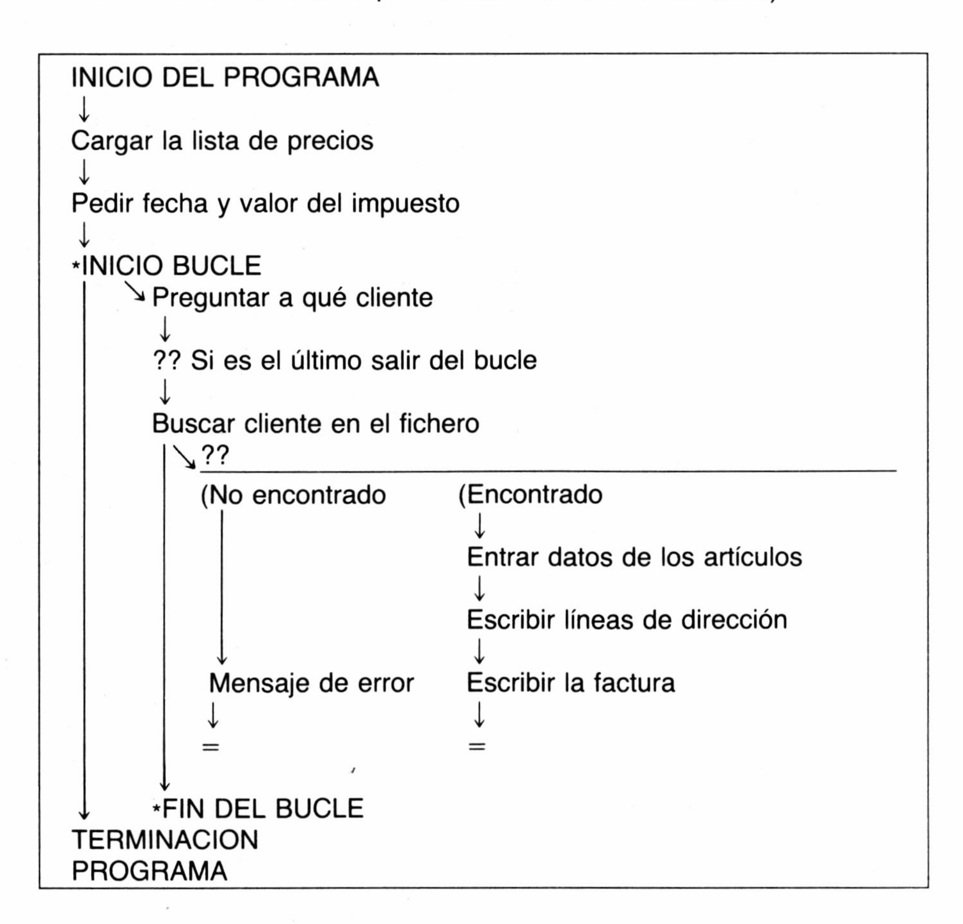
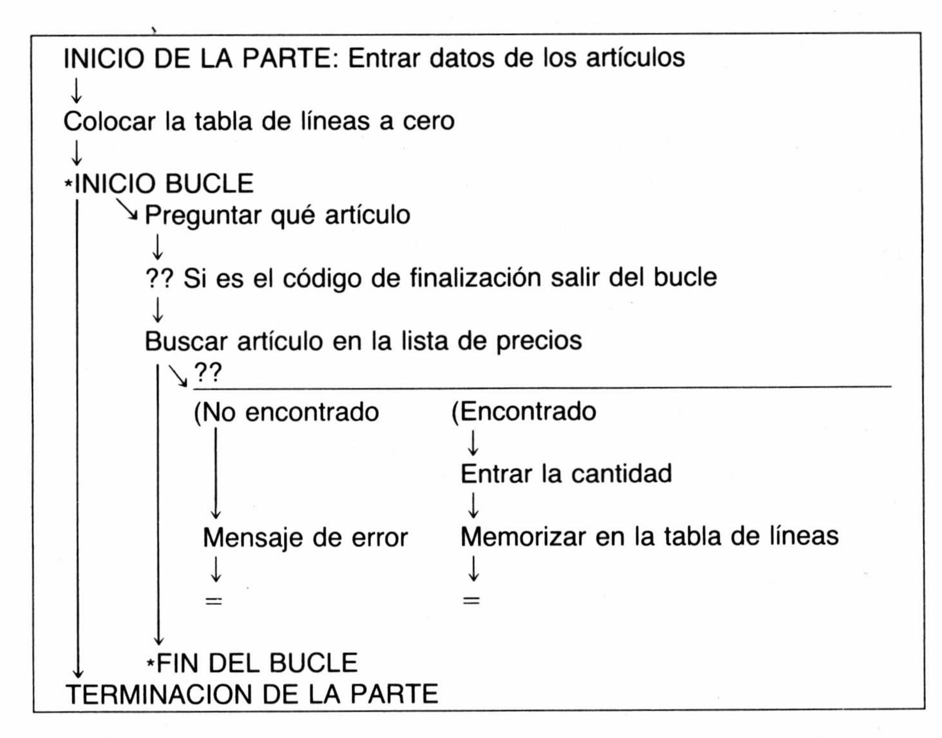


Figura 2. Esquema de funcionamiento del programa de facturación

Figura 3. Esquema del funcionamiento de la entrada de artículos



Si el resultado es encontrado, el programa ya puede escribir los datos del cliente que son las tres líneas de dirección.

Después debe pedir cuáles son los artículos que hay que facturar y qué cantidad, una vez terminada esta entrada de datos, se escribe la factura y se va a pedir un nuevo cliente al cual facturar.

Si el resultado es no encontrado, lo lógico es dar de alta al individuo, si no se trata realmente de una equivocación. Esta posibilidad existe en un disco magnético, en el fichero que disponemos en forma de DATA no es posible dar de alta. Es necesario antes de empezar a hacer facturas cambiar los DATAS con el fin de dar de alta o baja o modificar tanto el fichero de clientes como la lista de precios. Por lo tanto, si durante la ejecución de las facturas aparece un cliente que no está en el fichero se debe suponer que es un error. En este caso, se da un mensaje y se pide otro cliente.

Antes de seguir con la escritura del programa es necesario detallar un poco más cuál es la entrada de datos de los artículos.

La figura 3 muestra el esquema de esta parte.

Cuando se entra en esta parte se coloca en primer lugar la tabla de líneas a cero; la tabla de líneas consiste en una tabla que memoriza las líneas de artículos que debe aparecer en la factura.

A continuación se entra en un bucle que pide el código de un artículo; como en el caso anterior, existe un código, normalmente el cero, que indica que se ha finalizado y sirve para salir del bucle.

Si el código entrado no responde al código de final, entonces el sistema consulta la lista de precios para encontrar de qué código se trata.

El resultado de esta consulta puede ser código encontrado o no encontrado.

En el caso de código no encontrado el sistema nos da un error y nos pide otro código.

En el caso de que realmente se encuentre el código, nos pide la cantidad. Memoriza esta línea en la tabla y vuelve a pedirnos un nuevo artículo.

11.7.2.2 Estructura de datos

Para finalizar este diseño general es necesario detallar un poco más cuál es el significado, en cuanto a contenido, de la lista de precios, del fichero de clientes y de la tabla de líneas.

Empecemos por el fichero de clientes; el contenido es:

— Código del cliente. Es el elemento que sirve para buscar.

— Primera línea de la dirección.

— Segunda línea de la dirección.

— Tercera línea de la dirección.

A cada uno de estos elementos se les denomina en el lenguaje informático campos y al conjunto de los cuatro campos, registro del cliente. Pero sobre esta terminología ya hablaremos más adelante.

Respecto a la tabla de precios, los campos que precisamos son:

— Código del artículo. Es el elemento que sirve para buscar.

— Descripción del artículo.

— Precio del artículo.

La tabla de líneas consiste en dos campos:

— Subíndice de la tabla de la lista de precios. Observe que no es conveniente memorizar el código de artículo, pues cuando buscamos el código obtenemos un subíndice de la tabla de códigos de la lista de precios, que nos permite decir que el elemento está. Si memorizamos el código en el momento de escribir deberemos repetir la búsqueda otra vez con la consiguiente pérdida de tiempo y con unas instrucciones más complicadas.

— Cantidad que ha solicitado el cliente del artículo.

Después de esta descripción de los ficheros, reflexionemos qué estructura de datos nos conviene más.

En primer lugar, el fichero de clientes lo mantendremos en las instrucciones DATA para que la simulación del disco magnético sea lo más fiel posible. En general, una empresa dispone de un número de clientes tan elevado que no es posible mantenerlos todos en memoria. Pero nosotros tenemos que mantenerlos en DATA en nuestro programa. Evidentemente nos caben en memoria, pero se trata solamente de un ejemplo con un número reducido.

La lista de precios, en cambio, la mantendremos en tres listas que dimensionaremos a 10; es un ejemplo. Posteriormente ya daremos la dimensión que nos convenga. Por lo tanto, las dimensiones serán:

```
10 DIM c(10)      : REM Código de articulo
20 DIM a$(10,15): REM Descripción del artículo
30 DIM p(10)      : REM Precio del articulo
```

En la descripción del artículo se han reservado 15 caracteres por artículo, que es precisamente la longitud máxima que nos permite la escritura en la factura.

Normalmente las listas de precios caben todas en memoria, excepto en empresas en donde el número de artículos es muy elevado, por ejemplo, una ferretería. Insistimos que hemos tomado 10 de momento.

En un caso real la podemos aumentar a valores bastante más altos, siempre, claro está, que el ordenador tenga la memoria suficiente.

Finalmente la tabla de líneas es el siguiente par de listas:

```
40 DIM n(10) : REM Subíndice de la lista de precios.
50 DIM q(10) : REM Cantidad pedida del artículo.
```

Observe que también se dimensiona a 10. Evidentemente no se van a pedir más artículos que los que hay en la tabla de precios. Sin embargo, lo normal es que este número de líneas sea bastante menor que la lista de precios. El número de artículos que hay en un pedido es normalmente muy inferior al total de la lista de precios.

Puede parecer que esto es una seria limitación al programa, pero en realidad no lo es, pues si no cabe un pedido en una factura se hace más de una.

Finalmente el estudio de este esquema nos muestra el número y trozos de programas que deberemos confeccionar. La lista de actividades es la siguiente:

1. Cargar la lista de precios.
2. Entrada fecha e impuesto.
3. Fichero de clientes. Gestión completa excepto artículos.
4. Tabla de líneas.
5. Escritura de las líneas.

11.7.3 Escritura del programa

11.7.3.1 Carga de la lista de precios

El primer paso consiste en cargar la lista de precios que está colocada en DATA y es necesario pasarla a las tablas, ya diseñadas, para poder disponer de ellas con un acceso cómodo.

Como no sabemos cuantos precios tenemos, precisamos de un indicador de final para decirnos que se han terminado los datos relativos a la lista de precios. Se utiliza el código cero para indicar el final.

El trozo de programa que realiza la carga es:

```
100 REM Carga de la lista de precios.
110 LET i=1 : READ c(i),a$(i),p(i)
120 IF c(i) <= 0 THEN GO TO 150
130 LET i=i+1 : READ c(i),a$(i),p(i)
140 GO TO 120
150 LET na = i-1
```

La instrucción 110 coloca la variable *i* al valor 1, a continuación se leen con un READ los primeros elementos de las tablas de código de artículo, descripción del artículo y precio.

En la sentencia 120 se pregunta si el código es el indicador de final (se utiliza un valor negativo o cero); en el caso de que no lo sea, sigue por la línea 130, en donde, incrementa la variable *i* y lee otra vez de las instrucciones DATA. La línea 140 recicla el control de la pregunta de si hemos alcanzado el final.

Cuando la pregunta de que el código es cero o negativo es afirmativa, se envía el programa a la línea 150, en donde se calcula el número de artículo, que tenemos en la lista de precios.

Evidentemente, como se ha leído el indicador de final tenemos un elemento más en la tabla; por esto, se resta uno a la variable *i* y este valor se almacena en la variable *na*, número de artículos.

Para completar toda esta parte de programa es necesario disponer de los códigos de artículos en las instrucciones DATA.

Estas instrucciones son: (Para una lista de precios de ejemplo)

```
9000 DATA 101,"Papel",123
9010 DATA 102,"Lápices",14
9020 DATA 108,"Gomas de borrar",2
9030 DATA 105,"Tintero de tinta china roja",87
9040 DATA 000,"",0 : REM Fin de la lista de precios.
```

Tome nota de las consideraciones siguientes:

1. Los códigos aparecen desordenados, es la situación más normal en una lista de precios.

2. En la línea 9030 aparece la descripción del artículo muy larga. Esto no es ningún problema, pues el READ al leer la tabla *a$*, como en el programa le hemos fijado una longitud de 15, lo cortará adecuadamente. Quiere decir esto que cuando escribamos no será necesario preocuparnos por la longitud de las descripciones.

3. Como observa, la confección de los códigos de los artículos empieza en el 100. Esta manera es muy utiljzada en la codificación de cualquier tipo de artículos. Se evitan los números que tienen ceros a la izquierda y así siempre el código tiene un número de cifras concretas. Es decir, en este caso, deseamos un código de tres cifras, necesariamente debe haber

menos de 1000 productos, en general este tope es desmesurado. Si no lo es, se elige una cifra más.

Por lo tanto, podemos despreciar los códigos desde el 0 hasta el 99 que se pueden escribir con menos de tres cifras.

Si utilizamos un código de 4 cifras despreciamos los códigos desde el 0 hasta el 999 que se pueden escribir con menos de cuatro cifras.

Una vez tecleado este trozo de programa puede ejecutarlo con un RUN. El programa acaba sin hacer nada por pantalla. Para comprobar que funciona correctamente debe escribir las instrucciones inmediatas siguientes:

```
PRINT na
```

que debe dar cuatro, es el número de artículos que hay en la lista.

Luego haga

```
PRINT c(1),c(2),c(3),c(4)
```

deben aparecerle los cuatro códigos.

Luego haga

```
PRINT a$(1),a$(2),a$(3),a$(4)
```

deben aparecerle las descripciones de los artículos. Observe que el tintero de tinta china parece cortado, pues sólo se han previsto 15 caracteres para *a$*.

Finalmente haga

```
PRINT p(1),p(2),p(3),p(4)
```

debe salir los cuatro precios.

Si no obtiene estos resultados revise las instrucciones.

11.7.3.2 Entrada de la fecha y del impuesto

El segundo bloque de instrucciones a escribir es más sencillo. Se trata de la entrada de la fecha y del impuesto, que serán los mismos valores para todas las facturas que realicemos en la misma ejecución del programa.

El mecanismo es muy simple y se trata de dos instrucciones INPUT como se detallan a continuación:

```
200 REM Entrada de la fecha y del impuesto.
210 INPUT "Fecha:";f$
220 INPUT "Impuesto en % :";tpc
230 LET tpc = tpc/100
```

La variable que contiene la fecha se denomina *f$* y la que contiene el valor del impuesto se denomina *tpc*. La instrucción 230 lo único que hace es transformarnos la variable *tpc* de un significado de tanto por ciento a un significado de tanto por uno; este significado suele ser más cómodo para realizar los cálculos.

Puede probar ahora el funcionamiento de estos dos INPUT, ejecutando el programa.

11.7.3.3. Fichero de clientes

Esta parte del programa debe gestionar los clientes de los cuales queremos hacer la factura y es la traducción de la parte principal del esquema de la figura 2.

Las instrucciones son las siguientes:

```
 300 REM Bucle para cada uno de los clientes.
 310 INPUT "Código de Cliente:";x
 320 PRINT AT 20,1;"                                "
 330 IF x<=0 THEN GO TO 9999
 400 REM Búsqueda del código de cliente en el fichero.
 410 RESTORE 9100
 420 READ nc,l$,m$,n$
 430 IF nc = x AND nc>0 THEN GO TO 500
 440 IF nc > 0 THEN GO TO 420
 450 PRINT AT 20,1;"No se encuentra el código ";x
 460 GO TO 300
 500 REM Desarrollo del resto de la factura.
8000 GO TO 300
```

Se inicia esta parte preguntando el código del cliente del que queremos hacer la factura y se sitúa en la variable *x*.

La línea 320 nos escribe blancos en la línea 20 para colocar los mensajes de error a blanco, como veremos más adelante.

En la línea 330 se pregunta si el código es negativo o cero. En este caso se finaliza el programa de facturación, enviando el control a la última línea de programa; en el ZX-Spectrum es equivalente a la instrucción END.

La línea 410 rebobina las instrucciones DATA hasta la línea 9100 que es donde se deben iniciar los datos asociados al fichero de clientes.

La línea 420 lee los datos del código y tres variables textuales (l$, m$, n$) que contienen las tres líneas de dirección.

En la línea 430 se pregunta si el código coincide con el tecleado y además es mayor que cero. Tenga en cuenta que el cero es el marcador del final del fichero de clientes y, por lo tanto, si se pide el código cero en la variable x, esta instrucción nos encuentra igual el marcador de final y el

código pedido. La precaución de preguntar que además el código sea mayor que cero, impide este tipo de error.

Si la pregunta se contesta afirmativamente, es decir, el código leído y el código pedido coinciden, y además es mayor que cero, quiere decir que se ha encontrado el código del cliente y entonces se envía el control del programa a la línea 500, que es donde empezará la petición de los artículos.

En el caso de que la respuesta sea negativa se pasa a la lína 440, que pregunta si el código leído en el fichero es nulo. Si es cierto que aún no se ha alcanzado el final del fichero se recicla el programa a la línea 420 para leer los datos de otro cliente.

Si la pregunta es falsa sigue en la línea 450. En ella se imprime un error (el código de cliente no está en el fichero) y va a preguntar otro código de cliente, empezando en la línea 300, pues también hay que rebobinar el fichero al inicio del código de cliente para atender la pregunta de un nuevo código.

Finalmente se sitúa la lína 8000, porque cuando finalicemos la realización de una factura, enviamos el control del programa a pedir un nuevo código de cliente. La numeración de la línea se hace tan grande para estar seguros de que todos los bloques, que nos hacen falta para realizar la factura, caben en este intervalo de líneas.

Para complementar este bloque y poderlo probar es necesario colocar el fichero de clientes en los DATA oportunos.

Un fichero que puede servir como ejemplo es el siguiente:

```
9100 DATA 781,"La abeja","c/ Miel 23","08034 BARCELONA"
9110 DATA 654,"Los abetos","c/ Darwin 12","28022 MADRID"
9120 DATA 100,"El ocaso","Avda. de la Luz, 34","43012 VALENCIA"
9130 DATA 235,"Los hoyos","Céa Bermúdez, 12","28003 MADRID"
9140 DATA 102,"El americano","Ramblas 12","08001 BARCELONA"
9150 DATA 0,"","",""
```

en este fichero hay cinco clientes, el orden de la información es estrictamente tal como los lee la instrucción READ.

Observe que la línea 9150 contiene el marcador de final de fichero.

Puede ahora probar el programa hasta este punto mediante la introducción de los distintos códigos de clientes. No olvide probar algún código incorrecto para comprobar que el programa lo detecta.

Es aconsejable también interrumpir el programa con un STOP cuando pide el código y comprobar que las variables l$, m$ y n$ contienen los datos del cliente pedido.

11.7.3.4 Tabla de líneas

El objetivo de esta parte del programa consiste en memorizar el conjunto de artículos que debe contener la factura.

Como se ha indicado en el esquema de la figura 3, primero se coloca la tabla de memorización a cero.

De hecho no es necesario colocarla estrictamente todo a cero. Como se trata de una tabla para almacenar un número variable de líneas, deberemos tener una variable que nos lleve la cuenta exacta de líneas que hay que escribir en una factura. A esta variable la denominamos *lc*.

Para poner la tabla de líneas a acero basta con colocar la variable *lc* a cero, pues nos indica que la tabla contiene *lc* (ahora vale cero) líneas válidas.

Por lo demás, este trozo de programa es muy parecido al del apartado anterior pero en lugar de manipular los clientes manipulamos los artículos.

Por otra parte, una diferencia sustancial respecto al trozo anterior, es que tenemos la lista de precios en una tabla y, por lo tanto, el acceso es más inmediato que en el caso de las instrucciones DATA.

El programa de este trozo es el siguiente.

```
 500 LET lc = 0
 510 INPUT "Código de artículo:";x
 520 PRINT AT 20,1;"                                        "
 530 IF x<=0 THEN GO TO 1000
 600 FOR i = 1 TO na
 610 IF c(i) = x THEN GO TO 700
 620 NEXT i
 630 PRINT AT 20,1;"Código ";x;" incorrecto"
 640 GO TO 510
 700 REM se ha encontrado el código del artículo
 710 LET lc = lc + 1
 720 LET n(lc) = i
 730 INPUT "Cantidad:",q(lc)
 740 GO TO 510
1000 REM se inicia la escritura de la factura
```

Se inicia el bloque con la instrucción 500 que como ya hemos dicho coloca la variable *lc* a cero; es decir, número de líneas en la tabla igual a cero.

En la línea 510 se pregunta cuál es el código del artículo. Se utiliza la misma variable que para el código de cliente pues ya no se necesita, recuerde que los datos del cliente están situados en las variables l$, m$ y n$.

En la línea 520 se imprimen blancos para eliminar los errores.

La línea 530 pregunta si el código entrado es cero o negativo, este código indica que se ha terminado la entrada de los artículos y se puede pasar a escribir la factura. Se envía el programa a la línea 1000.

El bucle que se inicia en la línea 600 y termina en la 620 es el bucle de búsqueda del código de artículo en la lista de precios.

La variable que controla el final del bucle es la variable *na* que, en la primera fase de la escritura del programa, nos cuenta el número de elementos que hay en la lista de precios.

La línea 610 pregunta por la coincidencia de códigos, si esta es efectiva se envía el programa a la línea 700 para el procesado de esta línea ya que el código del artículo ha sido encontrado.

Si se termina el bucle y no se ha encontrado el código, el programa accede a las líneas 630 y 640 que consisten en informarnos del error y reciclar el programa a la entrada de un nuevo código de artículo.

Si hemos encontrado el código se accede a la línea 700, lo primero que se hace en la línea 710 es incrementar en uno el contador *lc* que nos da el número de líneas que hay que imprimir en la factura.

A continuación se memoriza en la lista *n(lc)* el valor de *i*, que es el subíndice de la lista de precios, que posteriormente no permite acceder a las informaciones de descripción y precio, sin necesidad de volver a realizar la búsqueda del código. Ya hemos mencionado antes que es mucho mejor almacenar este subíndice que el código, pues se evita otra vez la búsqueda cuando imprimimos.

Finalmente se pide la cantidad de artículo, que se coloca en la lista q(lc) que utilizaremos posteriormente en la impresión de la factura.

La línea 740 nos recicla el programa a la línea 510 para pedir un nuevo código del artículo.

Es conveniente hacer unas pruebas otra vez del funcionamiento del programa. Cuando se han entrado un par de artículos, compruebe en esta entrada que no se admiten los códigos incorrectos.

Una vez le pide otro código de cliente interrumpa el programa con un STOP y compruebe con PRINT directos el contenido de las variables.

Por ejemplo, si entra el código de artículo 108, en primer lugar y luego el código de artículo 102 y finalmente el 0. Debe obtener los resultados siguientes:

Al PRINT *lc* debe obtener un 2.

Al PRINT *q(1), q(2)* debe obtener las cantidades que ha entrado. Finalmente al PRINT *n(1), n(2)* debe obtener 3 y 2, ya que son el número de orden de los artículos en la lista de precios.

Efectivamente el tercero es el que tiene el código 108 que hemos entrado primero y el segundo es el que tiene el código 102.

Estudie bien este mecanismo de guardar los subíndices en lugar del código.

11.7.3.5 La escritura de la factura

En la fase alcanzada solamente es necesario realizar las instrucciones para escribir la factura en pantalla.

El programa es paralelo al desarrollo en la lección 5; de todas maneras volvemos a escribir aquí para aprovechar las nuevas instrucciones que conocemos.

El programa es el siguiente:

```
1010 CLS
1020 PRINT TAB(10);f$ : PRINT
1030 PRINT l$
1040 PRINT m$
1050 PRINT n$
1060 PRINT
1200 LET total = 0
1500 REM Impresion de las lineas de la factura.
1510 FOR i = 1 TO lc
1520 LET importe = p(n(i))*q(i)
1530 LET total = total + importe
1540 PRINT a$(n(i));" ";
1550 LET b$=STR$(p(n(i)))
1560 IF LEN b$ < 4 THEN LET b$=" "+b$ : GO TO 1560
1570 PRINT b$;" ";
1580 LET b$=STR$(q(i))
1590 IF LEN b$ < 4 THEN LET b$=" "+b$ : GO TO 1590
```

```
1600 PRINT b$;" ";
1610 LET b$= STR$(importe)
1620 IF LEN b$ < 5 THEN LET b$=" "+b$ :' GO TO 1620
1630 PRINT b$;" "
1640 NEXT i
1800 REM Escritura de los totales
1810 LET impuesto = INT ( total*tpc+0.5)
1820 PRINT TAB(26);"-----" : LET b$=STR$(total)
1830 IF LEN b$ < 5 THEN LET b$=" "+b$ : GO TO 1830
1840 PRINT TAB(19);"Total..";b$ : LET b$=STR$(impuesto)
1850 IF LEN b$ < 5 THEN LET b$=" "+b$ : GO TO 1850
1860 PRINT TAB(15);"Impuesto ";STR$(tpc*100);" ";b$
1870 LET total = total + impuesto
1880 PRINT TAB(26);"-----"
1890 LET b$ = STR$(total)
1900 IF LEN b$ < 5 THEN LET b$=" "+b$ : GO TO 1900
1910 PRINT TAB(26);b$
1920 PRINT TAB(26);"======"
```

La instrucción 1010 limpia la pantalla de la factura que tenemos en el caso anterior.

Las líneas 1020, 1030, 1040, 1050 y 1060 imprimen la fecha y las tres líneas de dirección del cliente y una línea de separación (1060). Recuerde que cuando hemos buscado en el fichero de clientes las tres líneas de dirección han quedado en las variables l$, m$, n$. La línea 1200 coloca la variable *total* a cero. Es la variable que nos sirve para ir sumando el valor total de la factura.

La línea 1510 abre un bucle con la instrucción FOR y la variable de control *i*. El bucle se cierra en la línea 1640. Este bucle recorre las líneas de artículos entradas anteriormente. La variable *lc* contiene el número de artículos entrados.

Cada vuelta dentro del bucle imprime una línea de la factura que contiene la descripción del artículo, el precio, la cantidad y el importe total de las piezas de este artículo.

Las líneas 1520 y 1530 calculan por una parte el importe asociado a esta línea y que se acumula en la variable *importe* y se acumula el total en la variable *total*.

Es necesario observar cómo se calcula el importe de la línea. El importe se calcula como el producto del precio por la cantidad.

La cantidad no ofrece ningún problema pues está almacenada en la tabla de líneas y se accede a ella mediante *q(i)*, es decir, especificando la cantidad asociada a la línea que se está escribiendo y que se memoriza en el programa en la variable de control del bucle.

El precio, sin embargo, no está en la tabla de líneas pues en ella sólo se ha memorizado el subíndice de la lista de precios que está contenida en *n(i)*. Entonces para acceder al precio hay que buscar en la tabla de precios en el índice que nos indica *n(i)*. Por esta razón aparece para el cálculo del precio *p(n(i))*. La tabla p contiene los precios. El subíndice está contenido en la tabla n, recuerde que después de la búsqueda sólo se almacena el índice del producto y no el código en la tabla de líneas.

La línea 1540 imprime la descripción del producto. La técnica que se utiliza para su acceso es la misma explicada en el caso de los precios. La

tabla n contiene el subíndice de la lista de precios que corresponde al producto.

Las líneas restantes se dividen en tres grupos, desde la 1550 hasta la 1570 constituye el primer grupo; desde la línea 1580 hasta la 1600 el segundo y desde la 1610 hasta la 1630 el tercero.

La función de cada grupo es idéntica; escribir una cantidad ajustada a la derecha; es decir, rellenada con blancos por la izquierda hasta un número determinado de caracteres.

Consideremos el primer grupo. La primera instrucción transforma una cantidad, en este caso el precio, a una cadena de caracteres que se denomina b$. La técnica para acceder al precio se ha explicado antes.

En la línea siguiente se alarga b$ hasta la longitud deseada, en este caso 4. El mecanismo consiste en preguntar si la longitud es menor que cuatro, si la respuesta es afirmativa se sigue por la instrucción THEN que añade un blanco a la izquierda de b$ y recicla el programa a la misma línea para volver a realizar la misma pregunta.

Finalmente la línea siguiente imprime la cadena b$ con un punto y coma al final para que el cursor no vaya a la línea siguiente. También se imprime un carácter blanco de separación.

El grupo siguiente realiza lo mismo pero para la cantidad pedida y el grupo siguiente para el importe de la compra de este artículo.

En este último grupo se elimina el punto y coma final pues ahora sí que se desea que se salte de línea (línea 1630).

Finalmente el grupo de líneas desde la 1800 hasta la 1920 se ocupan de escribir los totales de la factura.

Se utiliza la misma técnica que la explicada en la escritura de líneas para añadir los blancos a la izquierda para que los números queden ajustados a la izquierda.

La línea 1810 calcula el valor del impuesto con redondeo para eliminar los decimales. La técnica consiste en multiplicar el *total* por la del tanto por uno, *tpc*, y añadirle 0.5 unidades monetarias, a este valor se le quitan los decimales, de tal manera que si el valor del impuesto tiene unos decimales inferiores a 0.5 entonces se quitan, en el caso de que sea mayor toma la unidad siguiente.

Las demás instrucciones son para colocar los totales y el impuesto según especifica la maqueta de la factura de la figura 1.

11.7.4 Consideraciones finales

Aunque no se dé el listado completo del programa, las partes nos han permitido construir un programa bastante complejo.

En estos momentos debe Ud. probarlo muchas veces para ver que realmente funciona correctamente.

Además, puede añadir más clientes y más artículos para ver cómo es fácil poner y quitar diversos elementos, ya que los datos fijos están situados en instrucciones DATA. Observe que es bastante cómodo el hacer estas variaciones. Por otra parte, no olvide que si quiere tener una lista de precios de más de 10 artículos debe cambiar las dimensiones en las líneas 10, 20 y 30 que corresponden a la tabla de la lista de precios.

No parece necesario alargar la tabla de líneas ya que es difícil pensar en una factura de más de 10 líneas que tampoco nos caben en pantalla. En todo caso podemos hacer dos facturas como ya hemos mencionado antes.

Finalmente, volverle a aconsejar que estudie y repase bien esta práctica, pues contiene muchos elementos de funcionamiento de los ficheros que utilizan las grandes máquinas.

11.8 PRACTICA 2. DIBUJAR UNA FUNCION

11.8.1 Consideraciones generales

En esta práctica vamos a aprovechar el estudio de las funciones para dibujar cualquier función que se nos ocurra.

Realizar un dibujo en pantalla es parecido a hacer un dibujo sobre un papel cuadriculado normal. Como ya sabe el papel cuadriculado tiene una cuadrícula demasiado grande para que el dibujo quede bien. En la pantalla ocurre algo parecido. La diferencia mayor entre el papel y la pantalla es que en el papel utiliza las intersecciones de las líneas que delimitan la cuadrícula para representar los puntos, en cambio, en la pantalla se debe utilizar necesariamente el interior de los cuadrados.

En capítulos posteriores estudiaremos métodos más precisos; sin embargo, son más complicados y el método que se desarrolla aquí es de gran utilidad cuando sólo se quieren comportamientos generales más que valores muy precisos.

La cuadrícula de la pantalla contiene 32 × 20 cuadrados que corresponde a la intersección de 32 columnas y 20 líneas. La figura 4 muestra cómo se puede dibujar una línea recta en la pantalla o en el papel cuadriculado.

El eje vertical del papel se corresponde con las columnas de la pantalla (también puede ser al revés, es un convenio que establecemos ahora); en lenguaje matemático se denominan *ordenadas*.

El eje horizontal del papel se corresponde con las líneas de la pantalla; en el lenguaje matemático se denomina *abscisas*.

La manera de hacer el dibujo consiste, en primer lugar, en calcular para cada división horizontal el significado numérico que tiene; esto corresponde a definir la escala, término técnico que designa este proceso de asignación de número.

Como la pantalla, que es el equivalente de nuestro papel, tiene 20 líneas, sólo podremos colocar 20 marcas. Nos basta con saber el valor que debe asignarse a la primera línea y el incremento para obtener cuánto vale la abscisa en cuadrados siguientes.

Las marcas que se utilizan en la pantalla suelen ser o bien, la x o bien el símbolo +, ambos recuerdan un poco al aspa.

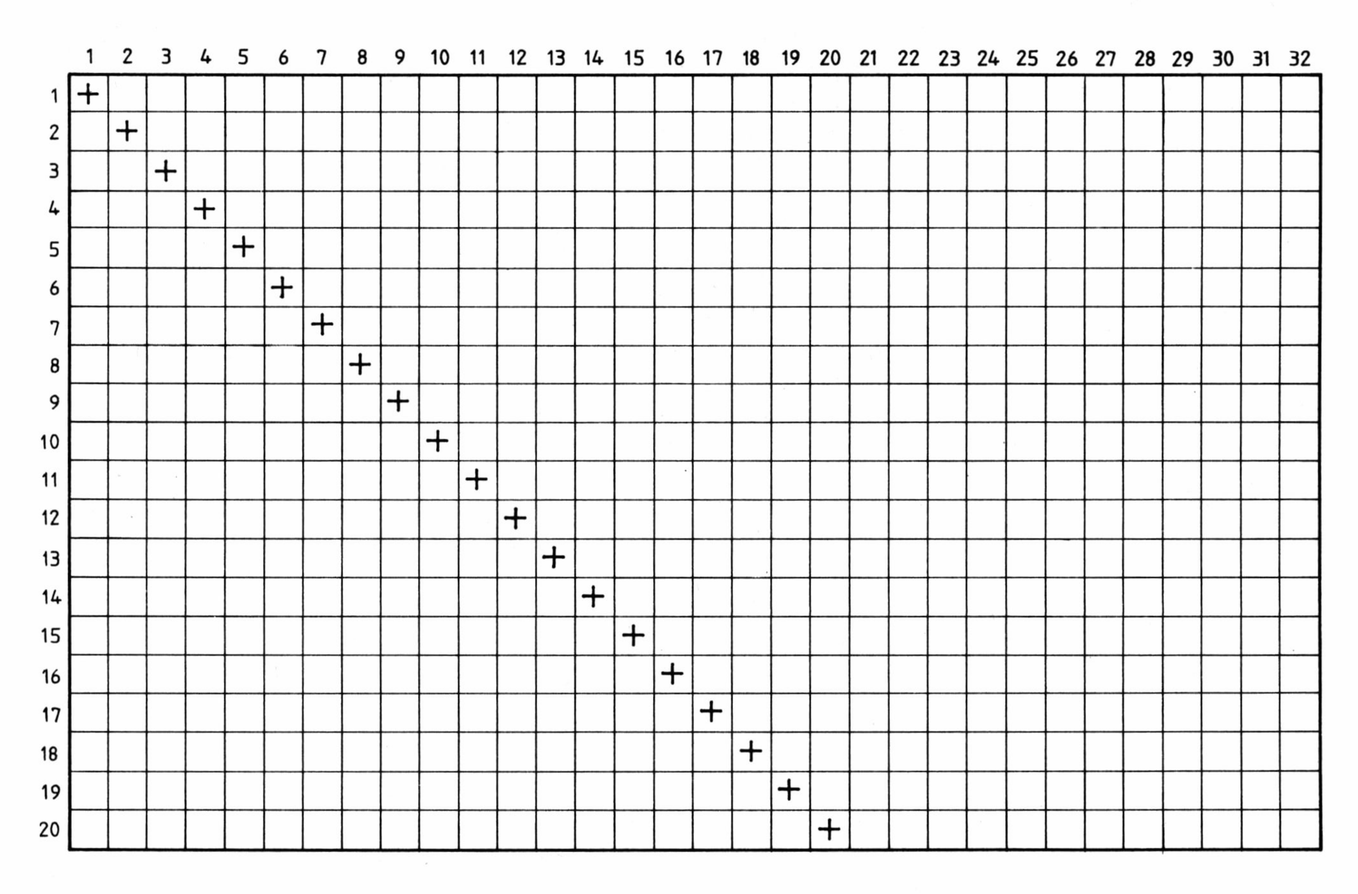


Figura 4. Representación de una línea recta en un papel cuadriculado

Una vez obtenido el valor que representa la línea, se calcula, mediante la función, la posición en que debe colocarse el aspa para que esté de acuerdo con el valor de la función.

11.8.2 Escritura del programa

Para empezar consideremos el ejemplo de la línea recta. En términos matemáticos una línea recta se representa mediante la función siguiente:

```
10 DEF FN f(x) = x
```

simplemente nos devuelve el valor del único argumento que tenemos.

Para representar esta función podemos utilizar el programa siguiente:

```
100 LET x = 0 : LET dx = 1
110 CLS
120 FOR i= 1 TO 20
130 PRINT TAB(FN f(x));"+"
140 LET x = x + dx
150 NEXT i
```

La instrucción 100 prepara justamente el valor de *x* con que se inicia el dibujo y el incremento que deseamos realizar en la variable *dx*.

A continuación se borra la pantalla y se inicia un bucle FOR en la línea 120, que se acaba en el NEXT de la línea 150.

En la línea 130 se imprime el aspa justamente detrás del valor del tabulador que marca la función. Vea que el argumento de la función TAB es precisamente el valor de la función que queremos dibujar.

En la línea 140 se incrementa el valor de *x* en la cantidad *dx* para preparar el valor de *x* asignado al siguiente cuadrado.

Por otra parte, el programa no confunde la *x* que utilizamos como variable con la *x* utilizada como argumento, tal como se ha explicado en el apartado dedicado a las funciones en esta lección.

Ejecute el programa y observe el resultado. Se traza una línea sobre toda la pantalla, tal como se ha dibujado en la figura 4.

La mejora más importante que se puede hacer en el programa es reciclar el control para que nos pregunte cada vez el valor inicial y el incremento de la función. De esta manera podremos seleccionar el intervalo del dibujo que deseamos.

Modifiquemos el programa así:

```
100 INPUT "Valor inicial:";x : INPUT "Incremento:";dx
105 IF dx = 0 THEN GO TO 9999
200 GO TO 100
```

La línea 100 nos pide ahora los valores de *x* y de *dx*. La línea 105 nos recoge la opción para finalizar, observe que cualquier valor de *dx* es bueno excepto el cero. En este caso no se incrementa para nada la variable dentro del bucle, el resultado es una línea vertical más o menos a la izquierda de la pantalla según el valor inicial seleccionado.

La línea 200 nos envía a una nueva pregunta de un valor inicial.

Ejecutamos ahora el programa con los datos siguientes:

1. $x = 0$ y $dx = 1$: El resultado debe ser el mismo que en el caso anterior.

2. $x = 5$ y $dx = 1$: La línea nos aparece desplazada hacia la derecha. El primer valor corresponde a la $x = 5$ y, por lo tanto, la función en la línea inicial vale 5.

3. $x = 30$ y $dx = -1$: Aparece una línea que va en sentido contrario, es decir, de izquierda a derecha; esto es debido a que hemos seleccionado un valor negativo del incremento. A medida que avanzamos en las líneas, el valor de *x* disminuye.

Esta prueba nos sugiere que se debe escribir el valor de las abscisas; es decir, cuánto vale la *x*, para tener idea clara de lo que se representa.

4. $x = 20$ y $dx = 1$: Aparece una línea partida en dos trozos. Esto es debido a que después de las doce primeras líneas la función vale 32 y la función TAB actúa dividiendo el valor por 32 y sacando el resto, es decir, lo sitúa en cero.

Este problema sugiere que se debe acotar a que el valor de la función

no supere el valor de 32. En el caso de que lo haga se imprime un valor distinto del signo + para indicar que la función sale de márgenes.

5. $x = -5$ y $dx = 1$: Aparece el error

B Integer out of range, 130:1

que nos indica que hay un valor incorrecto dentro de la función TAB, es negativo, de la línea 130.

Este problema sugiere una solución parecida al caso anterior, si la función es negativa se debe colocar un símbolo en cero para indicar que la función tiene un valor negativo.

Veamos a continuación cómo incorporar todas estas mejoras en nuestro programa.

En primer lugar veamos cómo se puede introducir el valor de la variable *x*. Lo haremos sólo cada cinco líneas para dar más claridad al dibujo.

Antes de imprimir el valor de TAB colocaremos una cadena de caracteres que representa el valor de *x*.

Las líneas a modificar son las siguientes:

```
121 LET a$ = "    "
122 IF i=1 OR i=6 OR i=11 OR i=16 THEN LET a$=STR$(x)
123 IF LEN a$ > 4 THEN LET a$ = a$(1 TO 4)
124 IF LEN a$ < 4 THEN LET a$ = " " + a$ :GO TO 124
125 PRINT a$;
130 PRINT TAB(FN f(x)+5);"+"
```

La línea 121 coloca en a$ cuatro blancos. En la línea 122 se pregunta si se trata de las líneas de pantalla 1, 6, 11 o 16, en este caso, se calcula a$ como la cadena que representa el número.

La línea 123 corta la cadena a$ a cuatro caracteres si es más larga que cuatro caracteres.

La línea 124 alarga la cadena si es menor que 4 caracteres hasta el valor de cuatro añadiendo blancos por la izquierda.

La línea 125 imprime el valor de a$.

Finalmente la línea 130 reajusta el valor del interior del TAB pues ahora el cero está situado en el sexto cuadrado del eje vertical. Se añaden 5 para dejar un espacio en blanco entre el valor de *x* y un posible valor de cero de la función.

Naturalmente ahora no disponemos de 32 columnas sino sólo de 27.

Ejecute el programa con los valores de $x = 0$ y $dx = 1$ y observe como aparecen los valores de la abscisa escrita cada cinco líneas.

Para solucionar los problemas de fuera de márgenes lo único que hay que hacer es calcular la función antes de imprimir y decidir si está el valor dentro de los márgenes de la pantalla. Si es así se imprime con la instrucción 130 y en caso contrario se escribe con una instrucción que imprima un símbolo distinto en los extremos de la pantalla. Se elige como símbolo el asterisco (*).

Las modificaciones que hay que hacer son las siguientes:

```
126 LET y = FN f(x)
127 IF y>0 THEN GO TO 129
128 PRINT " *" : GO TO 140
129 IF y>26 THEN GO TO 135
130 PRINT TAB(y+5);"+"
131 GO TO 140
135 PRINT TAB(31);"*"
```

La línea 126 calcula la función y la coloca en la variable *y*. La línea pregunta si la y es mayor que cero. En caso afirmativo envía el programa a la línea 129.

Si la respuesta es negativa, es decir, la *y* es menor que cero se imprime un blanco y un asterisco para denotar en el gráfico que el valor está por debajo de cero.

En la línea 129 se pregunta si la *y* es mayor que 26 que es el número máximo que podemos aceptar para la función TAB. Si la respuesta es negativa continúa el programa por la línea 130 que suma 5 al valor de *y*, para colocar el símbolo + en el lugar adecuado.

Cuando la respuesta es positiva se envía el programa a la línea 135 en donde se imprime un asterisco en el extremo derecho de la línea.

El programa completo queda así:

```
10 DEF FN f(x) = x
100 INPUT "Valor inicial:";x : INPUT "Incremento:";dx
105 IF dx = 0 THEN GO TO 9999
110 CLS
120 FOR i= 1 TO 20
121 LET a$ = "    "
122 IF i=1 OR i=6 OR i=11 OR i=16 THEN LET a$=STR$(x)
123 IF LEN a$ > 4 THEN LET a$ = a$(1 TO 4)
124 IF LEN a$ < 4 THEN LET a$ = " " + a$ :GO TO 124
125 PRINT a$;
126 LET y = FN f(x)
127 IF y>0 THEN GO TO 129
128 PRINT " *" : GO TO 140
129 IF y>26 THEN GO TO 135
130 PRINT TAB(y+5);"+"
131 GO TO 140
135 PRINT TAB(31);"*"
140 LET x = x + dx
150 NEXT i
200 GO TO 100
```

Realice ahora la prueba siguiente, dé a *x* el valor de -5 y a *dx* el valor 2. El dibujo de pantalla tiene una primera fase en la que aparecen asteriscos en la parte izquierda, denota que la función es negativa. Otra zona a partir de 26 aparecen asteriscos en la parte derecha, esto denota que la función es mayor que 26.

Después de estas modificaciones tenemos ya un programa que nos permite dibujar funciones con la seguridad de que se evitan errores de márgenes.

Finalmente pruebe con las funciones que le citamos a continuación y los intervalos que le damos.

1. Raíz Cuadrada: 10 DEF FN f(x) = SQR (x) *6

a) Valor inicial = 1; Incremento = 1

b) Valor inicial = 20; Incremento = 5

c) Valor inicial = 0; Incremento = 0,5

2. Interés Compuesto: 10 DEF FN f(x) = (1+x/100)ˆ5
   x define el interés y el número de años se considera fijo a 5 y se supone un capital de una unidad monetaria

a) Valor inicial = 1; Incremento = 3

b) Valor inicial = 25; Incremento = 0,5

Capítulo 12

ESQUEMA DE CONTENIDO

12.1 OBSERVACIONES GENERALES

El capítulo que inicia es el que contiene más elementos comunes en todas las versiones BASIC. En otras palabras, las diferencias entre las distintas versiones de BASIC prácticamente no existen.

De todas maneras, queremos recordarle antes de empezar algunas características que propiamente no son de la lección, pero que debe tener en cuenta para que cuando haga los programas en el ZX-Spectrum no tenga inconvenientes.

De hecho, le recordamos unas cuantas cosas que se han estudiado en capítulos anteriores.

Finalización del programa

En primer lugar en muchos programas se finaliza con un END. Ud. no dispone de esta instrucción en el ZX-Spectrum, pero puede sustituirla por un STOP o bien por un GO TO 9999.

Verá que la utilización del END en este capítulo es imprescindible, pues es necesario dividir el texto del programa en una parte que denominaremos programa principal y en otra que llamamos subrutinas. Muchos de los programas que se han hecho en capítulos anteriores finalizaban en la última línea. Por lo tanto, la utilización de STOP o el GOTO 9999 era innecesaria. En cambio, en este capítulo no ocurre lo mismo.

Programa de alineado de números

En segundo lugar aparece la función LEFT$ en la última modificación de la instrucción 530 del programa de *alineado de números por la derecha*.

En el caso del ZX-Spectrum debe sustituirse por el operador de fragmentación siguiente:

```
530 IF L>N THEN LET A$="*"+A$(1 TO N-1) : RETURN
```

que equivale a tomar los N-1 caracteres izquierdos de A$.

También aparece la función MID$ en la instrucción

```
630 LET X$ = MID$(A$,L-K+1,1)+X$
```

Acceso subrutina a subrutina

en el apartado correspondiente *al acceso subrutina a subrutina*. Recuerde que en el ZX-Spectrum no existe esta función y, en cambio, existe el operador de fragmentación.

Debe escribir esta instrucción como:

```
630 LET X$ = A$(L-K+1 TO L-K+1) + X$
```

Finalmente se utiliza también la función INKEY$, que el ZX-Spectrum sí posee, pero es más conveniente utilizarla con un PAUSE 0 delante para evitar la repetición de la tecla.

Programa listados

En el programa del apartado *listados* deben sustituirse las líneas 420 y 430 por

```
420 PAUSE 0 : LET A$=INKEY$
430 Debe eliminarse
```

Para cerrar esta parte de observaciones, le indicamos que en el ZX-Spectrum el error que da si se encuentra una instrucción RETURN sin tener una subrutina pendiente es:

7 RETURN without GOSUB 0:1

que significa que hay un RETURN sin el correspondiente GOSUB.



12.2 UN EDITOR DE TABLA

Seguramente le sorprende el título de este apartado. La palabra editor la conoce pero probablemente no con la acepción que se le quiere dar aquí. Le podemos decir que, incluso, esta acepción no estamos seguros de que sea correcta en castellano.

En informática la palabra editor se da a los programas que permiten escribir, modificar o borrar información que hay en la memoria del ordenador.

Cuando nos referimos a memoria nos referimos a una parte de memoria en la que tenemos control, porque pertenece a algún dato del programa.

De hecho, los editores en general son herramientas imprescindibles para la entrada de información en los ordenadores.

Ya conoce el funcionamiento de un editor de líneas: ¿sabe cuál? En el ZX-Spectrum, todo lo que puede hacer Ud. con las flechas, o la tecla de borrar (delete), o cuando entra una línea de programa, que no es más que un editor de las líneas. Recuerde que la tecla que permite modificar una línea se llama EDIT (editor).

El objetivo del resto de este capítulo es construir un editor para unos datos que están en una tabla.

Toda tabla tiene unas filas y unas columnas que deseamos rellenar con números. Un proceso posible es ir pidiendo el valor de cada una de las filas y columnas en un orden determinado. Este tipo de proceso es pobre e incómodo. Cuando se ha entrado un número no es posible retocarlo y cuando la tabla es grande la posibilidad de cometer errores aumenta notablemente. Por otra parte, si la tabla no está bien rellenada, el resultado del proceso puede ser erróneo y hay que volver a empezar a entrar datos y posiblemente cometer nuevos errores.

En procesos de entrada de datos largos es necesario buscar la máxima flexibilidad para que el usuario se sienta cómodo frente a los errores que pueda cometer.

Como el objetivo es ambicioso, vamos a abordar la construcción de un editor de tabla en cuatro etapas:

1. Visualización de la tabla.
2. Movientos de situación.
3. Entrada de datos.
4. Guardar estos datos en un cassette.

Cada una de estas partes es objeto de una práctica independiente (o casi independiente) de las demás. La potencia que tienen las subrutinas para descomponer el problema permiten esta independencia, pues el mecanismo básico es añadir al programa anterior las operaciones (son subrutinas) que nos faltan para conseguir el objetivo parcial de la etapa.

De todas maneras, es necesario definir el problema ahora de una manera general para dejar claro el objetivo que deseamos alcanzar. Debido a que el programa es largo prepare el cassette y una cinta nueva para guardar las diversas versiones y partes del programa.

12.2.1 Definición del problema

Se trata de realizar un programa que permita la entrada y la modificación de los valores de una tabla de 30 filas y 15 columnas y que cada intersección de una fila y una columna tenga 7 caracteres.

En cada casilla sólo se pueden entrar números enteros positivos, es decir, sin ningún decimal ni signo.

En términos generales se trata de mantener en pantalla el máximo de casillas y mediante las flechas poder «pasearse» por cada una de las casillas y modificar el resultado si hace falta en cada una de ellas.

Un pequeño cálculo nos muestra que una tabla de estas características no cabe en la pantalla. En efecto, en el ZX-Spectrum tenemos 32 columnas y se requieren 15 columnas de 7 caracteres que dan un total de 105 columnas. Por otra parte disponemos de 20 filas y la tabla tiene 30.

El mecanismo de inspección de la tabla se construye como si la pantalla fuera un visor de un tamaño determinado y la tabla un papel más grande. El mecanismo de «paseo» consiste en poder mover el visor sobre el papel, de tal manera que podemos ir viendo el trozo de tabla que queramos.

A medida que inspeccionamos la tabla podemos modificar los valores de las intersecciones de las filas y las columnas para colocar o modificar los valores actuales.

Una vez realizado este proceso de edición guardaremos el trabajo realizado en un medio magnético (la cinta de cassette) para poder seguir más tarde o aprovecharlo para que otros programas realicen cálculos con esta tabla.

12.3 PRACTICA 1. VISUALIZACION DE LA TABLA

12.3.1 Consideraciones iniciales

La tabla de datos es lo primero que necesitamos para poder rellenarla con algo. Luego comenzaremos nuestro programa por reservar la memoria necesaria para tener la tabla que contiene los datos.

También agrupamos todas las constantes del programa, antes de realizar ninguna acción, para poder variar estos datos con facilidad, modificando únicamente esta primera parte del programa. Es decir, partimos del supuesto de una tabla de 30 filas y 15 columnas, pero es posible que en otra ocasión necesitemos otro número de filas y de columnas. Por lo tanto, es mejor dejar estos números como variables del programa, que se asignan al iniciarse el programa.

Por ello, las primeras instrucciones agrupan todas estas constantes que a lo largo del diseño y escritura necesitemos.

Como la práctica es larga le recomendamos que en un papel aparte se vaya haciendo una lista de las constantes y variables que utilizamos, indicando lo que representan.

Las primeras líneas son definir precisamente el tamaño de la tabla a utilizar.

```
10 REM Inicialización de las constantes del programa.
20 LET tf = 30 : LET tc= 15
30 LET ac = 7
```

La constante *tf* indica el número de filas y la constante *tc* indica el número de columnas. Como la tabla es de caracteres, en el ZX-Spectrum es necesario especificar cuántos caracteres debe tener; por lo tanto, se define *ac* como el ancho de caracteres que tiene la tabla.

Como se precisarán más constantes, dejamos un espacio de líneas para ir agrupándolas todas al principio del programa.

La instrucción siguiente define la tabla que utilizamos para almacenar los datos.

```
100 REM Dimensionado de las tablas.
110 DIM t$(tf,tc,ac)
```

Para poder iniciar el trabajo rellenamos esta tabla con algún valor que permita controlar que estamos en el camino correcto en las pruebas del programa. Por ejemplo, se puede rellenar la tabla con una cadena de caracteres que indique a qué fila y columna pertenece el elemento. Más tarde, cuando el programa esté probado con esta tabla, sustituiremos estos valores por los requeridos para las otras fases.

```
500 REM Llenado inicial de la tabla.
510 FOR i = 1 TO tf
```

```
520 FOR j = 1 TO tc
530 LET t$(i,j) = STR$(i)+","+STR$(j)
540 NEXT j
550 PRINT i, : NEXT i
```

Las instrucciones consisten en dos bucles: uno que recorre las filas (variable de control *i*) y otro que recorre las columnas (variable de control *j*). En el interior de estos bucles anidados se construye el valor de la tabla mediante la concatenación de la cadena de caracteres que representa *i*, una coma y la cadena de caracteres que representa la *j*.

En la línea 550, antes del NEXT i, se escribe qué línea se rellena, para poder controlar el progreso del programa. Este control es necesario, pues se tarda bastante en rellenar la tabla si es grande. En nuestro supuesto hay que rellenar 450 elementos (30 × 15).

En este momento ya se puede probar el programa en esta primera fase. Escriba el RUN y a la tecla de fin de línea.

En pantalla aparecen dos columnas: la de la izquierda escribe que se han llenado las filas impares y la derecha escribe que se han llenado las filas pares. La coma detrás del PRINT de la línea 550 tiene esta misión.

Una vez finalizado el programa debe comprobar que los elementos de *t$* contienen los datos que deseamos. Para ello utilice las instrucciones de ejecución inmediata. Por ejemplo, escriba

```
PRINT t$(6,12)
```

en pantalla debe aparecer 6,12. Es decir, tal como se ha llenado la tabla la impresión de un elemento nos debe dar los índices de este elemento.

Haga unas cuantas pruebas de este tipo para diferentes combinaciones de índices para ver si la tabla se ha llenado según lo que se ha planificado.

12.3.2 Descripción del visor

El siguiente punto a abordar es la visualización de la tabla en pantalla. Realmente la tabla entera no cabe en la pantalla. Por ello utilizamos la pantalla para ver una parte de la tabla de datos.

Con las características del ZX-Spectrum, que dispone de 32 columnas y 20 filas, caben en pantalla 20 filas de cuatro datos de una anchura de ocho columnas. Si se han de añadir las cabeceras para tener la indicación de las filas y columnas en que estamos, la anchura apropiada para los datos es de 7 caracteres tal como hemos definido más arriba.

Con cuatro datos de 7 caracteres cubrimos 28 columnas, las cuatro restantes las utilizamos para indicar en qué fila estamos. Esta constante de cuatro columnas se debe incluir en la definición de las constantes mencionada más arriba.

La figura 1 nos muestra la estructura de pantalla que deseamos.

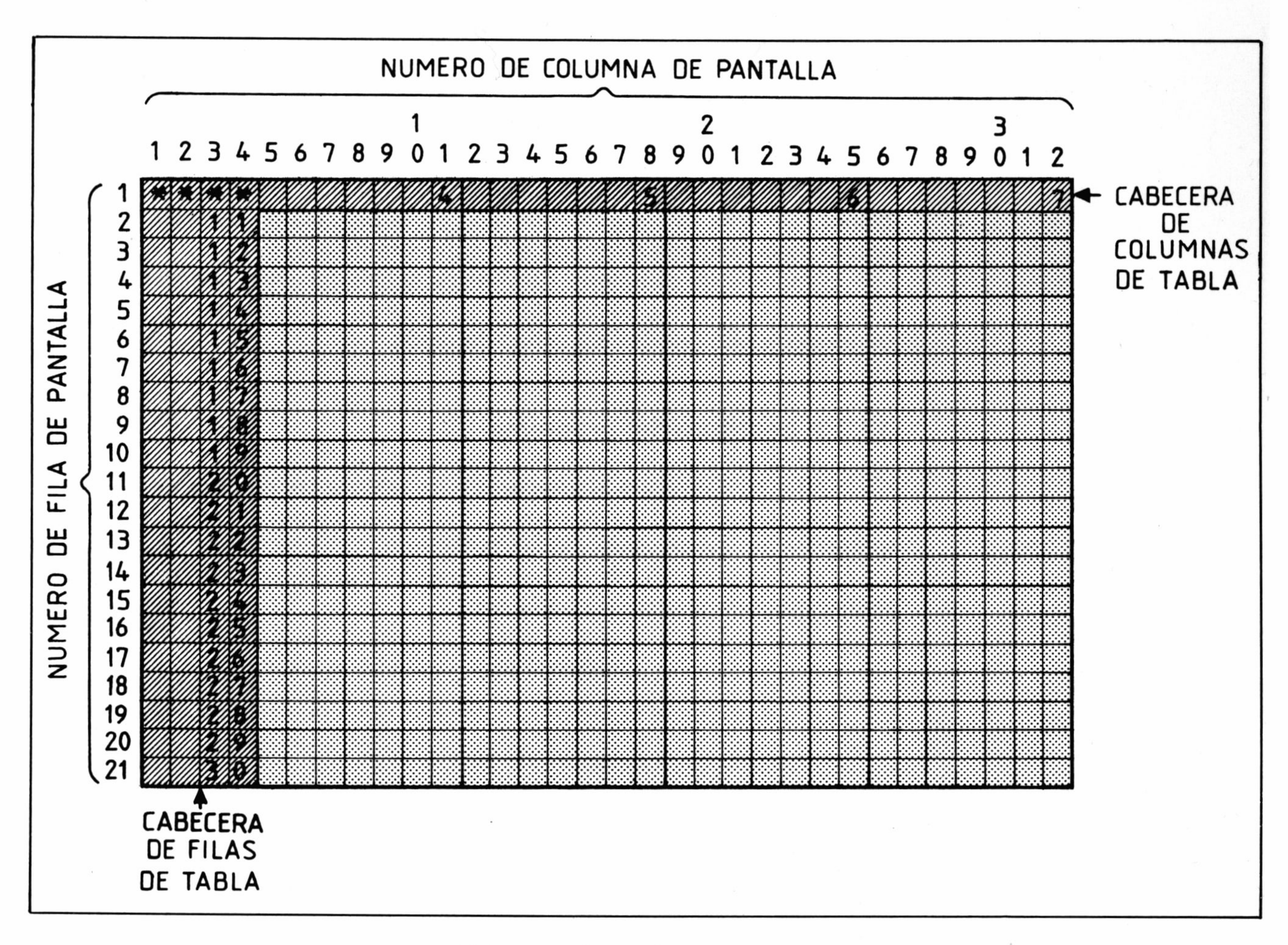


Figura 1 Esquema de visualización del visor de pantalla. La parte tramada corresponde al diseño de la pantalla. Se muestra el caso de que el visor esté situado en la fila 11 y en la columna 4 de la tabla.

Esta pantalla se denomina visor porque sólo puede visualizar 4 columnas y 20 filas. Para ver distintas partes de la tabla utilizamos el movimiento de esta pantalla sobre la tabla. Para hacerlo se selecciona la fila y columna inicial que deseamos visualizar en la pantalla, que corresponde a la casilla superior izquierda.

Además, para distinguir claramente lo que es contenido de lo que es información de situación se colocan en modo inverso (el fondo negro y las letras blancas) los datos informativos (en la figura aparece tramado).

Es adecuado construir la impresión del visor con una subrutina, pues hay que utilizarla muchas veces para poder realizar el «paseo» a lo largo de la tabla.

Los datos que se deben dar a esta subrutina son qué fila y qué columna se deben visualizar en la parte superior izquierda de la pantalla. Por ejemplo, en la figura 1 se ha indicado la fila 11 y la columna 4.

Para simplificar al máximo el proceso de escritura construimos estas cabeceras en unas listas ya preparadas para el proceso de impresión. Tenga en cuenta que el movimiento del visor debe agilizarse al máximo para que la respuesta del programa no sea excesivamente lenta.

A estas cabeceras las denominamos *f$* y *c$*, indicaciones de filas y

columnas respectivamente. Las dimensiones son para *f$* la de las filas de la tabla y para *c$* las de las columnas de la tabla.

Observe, empero, que en la parte superior izquierda de la pantalla queda un hueco que corresponde a la intersección de la cabecera de filas y la cabecera de columnas. En la figura se señala con asteriscos.

Es cómodo colocar una cabecera de fila más con este espacio en blanco. De esta manera la impresión de la pantalla no necesita saber el ancho de la columna dedicada a la cabecera de filas.

El número de caracteres de *f$* es de cuatro, tal como hemos razonado más arriba. El número de caracteres de *c$* es de 7. Debe tener exactamente el mismo ancho que las columnas de datos.

12.3.3 Escritura del programa

La figura 2 muestra el programa para realizar las pruebas de la impresión del visor en pantalla. No escriba todavía esta parte del programa. Vaya siguiendo las explicaciones sobre la figura 2.

Las líneas 10, 20, 30 y 40 corresponden a la inicialización de las constantes. Se han añadido las variables

tpf que indica el número de filas que tiene el visor.

tpc que indica el número de columnas que tiene el visor.

acb ancho de la cabecera de filas.

Las líneas 100, 110 y 120 son el dimensionado de las tablas. Los nombres son:

t$ tabla a editar propiamente dicha.

f$ lista de la cabecera de filas.

c$ lista de cabecera de columnas.

Observe que el dimensionado se hace a *ac* caracteres para la tabla y la cabecera de columnas; a *acb* caracteres para la cabecera de filas.

También la cabecera de filas se ha dimensionado a *tf* + 1 para incluir el elemento intersección de la cabecera de columna con la cabecera de filas.

De la línea 220 hasta la 310 se rellenan las listas de las cabeceras. En primer lugar, los números nos quedan mejor situados en una tabla si se ajustan a la derecha. Para realizar esta operación se ha diseñado la rutina 9900 que ajusta un texto a la derecha.

Figura 2 Programa para las pruebas de visualización del visor, que se inicia en la fila q *y en la columna* p.

```
10 REM Inicialización parámetros.
20 LET tf = 30 : LET tc= 15
30 LET tpf = 20 : LET tpc = 4
```

```
  40 LET ac = 7 : LET acb = 4
 100 REM Dimensionado tablas.
 110 DIM t$(tf,tc,ac) : DIM c$(tc,ac)
 120 DIM f$(tf+1,acb)
 200 REM Rellenado tablas
 210 REM Cabecera de las filas
 220 LET a = acb
 230 FOR i = 1 TO tf
 240 LET a$ = STR$(i) : GO SUB 9900 : LET f$(i)=a$
 250 NEXT i
 260 LET a$ = "" : GO SUB 9900 : LET f$(tf+1)=a$
 270 REM Cabecera de columnas
 280 LET a = ac
 290 FOR i = 1 TO tc
 300 LET a$ = STR$(i) : GO SUB 9900 : LET c$(i) = a$
 310 NEXT i
 500 REM Rellenado provisional de la tabla
 510 FOR i = 1 TO tf
 520 FOR j = 1 TO tc
 530 LET a$ = STR$(i)+","+STR$(j) : GO SUB 9900
 540 LET t$(i,j) = a$
 550 NEXT j
 560 PRINT i,
 570 NEXT i
 580 PRINT
1000 REM Prueba del funcionamiento.
1010 INPUT "Fila inicial:";q
1020 INPUT "Columna inicial:";p
1030 GO SUB 9500
1040 GO TO 1000
9500 REM Escritura de una pantalla que se inicia en q,p
9510 PRINT AT 0,0;"«";f$(tf+1);"»"; : REM primera línea.
9520 FOR l = p TO p + tpc-1
9530 PRINT "«";c$(l);"»";
9540 NEXT l
9550 PRINT
9600 FOR l = q TO q +tpf-1 : REM Líneas sucesivas
9610 PRINT "«";f$(l);"»";
9620 FOR k = p TO p +tpc-1
9630 PRINT t$(l,k);
9640 NEXT k
9650 PRINT
9660 NEXT l
9670 RETURN
9900 REM Subrutina de ajuste a la derecha.
9910 IF LEN (a$)> a THEN LET a$ = a$(1 TO a)
9920 IF LEN (a$) <a THEN LET a$ = " "+a$ : GO TO 9920
9930 RETURN
```

12.3.3.1 Rutina de ajuste a la derecha

Esta rutina requiere dos argumentos: un texto (*a$*) y una longitud para añadir blancos por la izquierda (*a*).

La línea 9910 corta la cadena de caracteres *a$* a *a* si sobrepasa el valor definido. En este tipo de programa esto no tiene utilidad, pues la idea es que la subrutina devuelva un texto de una longitud *a* con caracteres en blanco rellenando la parte izquierda si el texto tiene inicialmente una longitud menor que *a*.

La línea 9920 añade caracteres en blanco a la izquierda de *a$* hasta alcanzar la longitud especificada por *a*.

12.3.3.2 Relleno de las tablas

La línea 220 coloca el valor de *a* que se utiliza en la subrutina a la longitud del ancho de cabecera de las filas que está en la variable *acb*.

Se inicia un bucle en la línea 230 que recorre todas las filas. La línea 240 asigna a *a$* el valor en caracteres de la fila en que se encuentra (variable *i*) y se envía a la subrutina de ajuste a la derecha. Una vez retorna se coloca en el elemento de lista correspondiente.

Al finalizar el bucle se ejecuta la línea 260, en donde el valor de a$ se rellena con blancos y se asigna al elemento adicional de la cabecera de filas, para tener en cuenta la intersección con la cabecera de columnas.

Las líneas 270 hasta la 310 hacen el mismo trabajo para la cabecera de columnas; el único detalle a tener en cuenta es que en la línea 280 se fija la longitud del elemento a *ac*, que es el ancho de columna de datos.

Las líneas 500 hasta la 580 realizan la misión de colocar en la tabla de datos una cadena de caracteres con la fila a que pertenece, una coma y la columna a que pertenece. Respecto a la versión anterior se ha añadido el proceso de ajustar a la derecha.

12.3.3.3 Impresión del visor

De la línea 1000 hasta la 1040 está el bucle para comprobar el funcionamiento. Los datos que se entran son la fila y la columna que inician la parte izquierda superior del visor.

A continuación en la línea 1030 se envía a la subrutina de visualización del visor.

La línea 1040 recicla el proceso.

La subrutina de visualización del visor se inicia en la línea 9500.

La línea 9510 sitúa el inicio de la impresión en la parte superior a la izquierda de la pantalla, mediante la posición del cursor (función AT). A continuación se escribe el elemento *f$(tf+1)*, que es justamente el elemento que se ha preparado para imprimir la intersección de la cabecera de filas con la cabecera de columnas.

Observe que en esta línea aparece una indicación ⋞ y una indicación ⋟ . Estos caracteres no se encuentran en el ZX-Spectrum; los colocamos para indicarle que se desea esta parte escrita en fondo inverso. Es decir, letras blancas sobre fondo negro.

El mecanismo para escribir esta indicación es el siguiente: escriba la línea tal como se la mostramos, pero sin pulsar el signo que hay entre las comillas. Sí debe pulsar en cambio las comillas; es decir, escriba una cadena vacía.

A continuación edite esta línea con la tecla EDIT y sitúe el cursor justo entre las dos primeras comillas y pulse la tecla INVERSE VIDEO. El resto de la línea se coloca en fondo negro y letras blancas. Utilice las teclas del cursor para situarse entre las comillas finales y pulse la tecla de TRUE VIDEO, el resto de la línea se coloca en forma normal. Finalice la edición de la línea mediante la tecla de fin de línea (ENTER).

Las líneas 9520, 9530 y 9540 consisten en un bucle para escribir las cabeceras de las columnas. La línea 9530 utiliza también las indicaciones mencionadas para invertir el fondo y las letras. Utilice el mismo mecanismo explicado más arriba. (Ojo en estas líneas y las siguientes en no confundir la ele minúscula con el número 1.)

Observe que como el visor debe iniciarse en la columna *p*, el bucle de cabeceras de columna se inicia en *p* y se acaba en *p + tpc −1*, pues la pantalla sólo puede visualizar correctamente *tpc* columnas.

El menos 1 se coloca porque si caben 4 columnas y empezamos en la 5; la 9 ya no debe visualizarse. En efecto, se visualizan 5, 6, 7 y 8.

Desde la línea 9600 hasta la línea 9650 se realiza un bucle para escribir el resto de filas del visor. Los límites del bucle van desde *g* hasta *g+tpf −1* por la misma razón que se ha explicado en las columnas.

En el interior del bucle hay que escribir una línea cada vez. La línea 9610 escribe la cabecera de la fila con el fondo invertido se aplica la misma técnica mencionada en el caso de la cabecera de columnas. Observe que se escribe *f$(1)*, que corresponde a la cabecera de la fila *l*(ele), que es la que se imprime y está contenida en la variable de control del bucle.

La impresión se acaba en punto y coma, porque aún no se ha acabado la línea.

En la línea 9620 se inicia el bucle para escribir las columnas propiamente dichas. Los límites del bucle van desde *p* hasta *p+tpc−1* por la misma razón explicada en la cabecera.

El interior del bucle consiste en una sencilla instrucción PRINT que imprime el elemento de índice *l* de filas (la variable de control del bucle externo) y *k* de columnas (la variable de control del bucle más interno).

La instrucción finaliza con un punto y coma para no saltar de línea de la pantalla.

La línea 9640 finaliza el bucle de las columnas y la línea 9660 realiza un PRINT para saltar a la línea siguiente.

Una vez finalizado el bucle que acaba en la línea 9660 se ha impreso toda la pantalla.

La línea 9670 retorna el control al programa principal.

Ahora ya puede escribir el programa de la figura 2, siguiendo las indicaciones dadas.

12.3.4 Prueba del programa

Ya se puede ejecutar el programa que empieza mostrando cómo va inicializando las tablas de los datos.

Una vez se ha terminado nos pide la fila inicial y la columna inicial.

En primer lugar pediremos la fila 1(ENTER) y la columna 1(ENTER). En pantalla deben aparecer 20 filas con los índices del 1 al 20 y cuatro columnas con el índice del 1 al 4.

Una vez realizado esto correctamente (si no es así repase la subrutina de impresión) damos como dato la fila 5 y la columna 5.

En pantalla debe aparecer la tabla que abarca desde las filas 5 hasta la 24 y las columnas 5 hasta la 8.

Finalmente, pruebe desde la fila 15 y con la columna 8. La visualización se inicia normalmente. Se muestran las columnas 8, 9, 10 y 11, para las filas 15 y sucesivas. En el momento de escribir la fila 31, cuya cabecera es un blanco (recuerde que hemos utilizado este elemento paa escribir la intersección de las cabeceras de filas y columnas), no escribe el 31 sino que aparece un error que nos indica que hemos sobrepasado el tamaño de la tabla.

Este error es previsible y lógico, pues no hemos colocado ningún mecanismo de control de si el visor sale o no fuera del papel. Este mecanismo no hace falta colocarlo en la subrutina de impresión, pero debe tenerse en cuenta cuando realicemos la parte del programa relacionada con el paseo del visor sobre la tabla de datos.

Una vez probado el programa puede grabarlo en el cassette con el nombre «p1».

12.4 PRACTICA 2. MOVIMENTOS DE SITUACION

12.4.1 El cursor y sus movimientos

En la práctica anterior hemos definido la pantalla como el visor de una tabla mucho mayor.

El objetivo último es poder acceder a cada uno de los elementos de la tabla para poder cambiarlos de valor de acuerdo con unos datos que debemos memorizar.

Es necesario, por lo tanto, disponer de un mecanismo que nos indique en qué elemento de la tabla se trabaja. A este indicador se le denomina *cursor*.

En el programa señalamos el cursor mediante la impresión del elemento en video inverso, es decir, fondo negro y letras blancas.

Una vez establecido el mecanismo de señalización hay que considerar el movimiento del cursor a lo largo y a lo ancho de la tabla.

Hay cuatro movimientos básicos de este indicador:

— Arriba : Se sitúa en el elemento que está en la misma columna y en una fila menos.

— Abajo : Se sitúa en el elemento que está en la misma columna y en una fila más.

— Izquierda : Se sitúa en el elemento que está en la misma fila y en una columna menos.

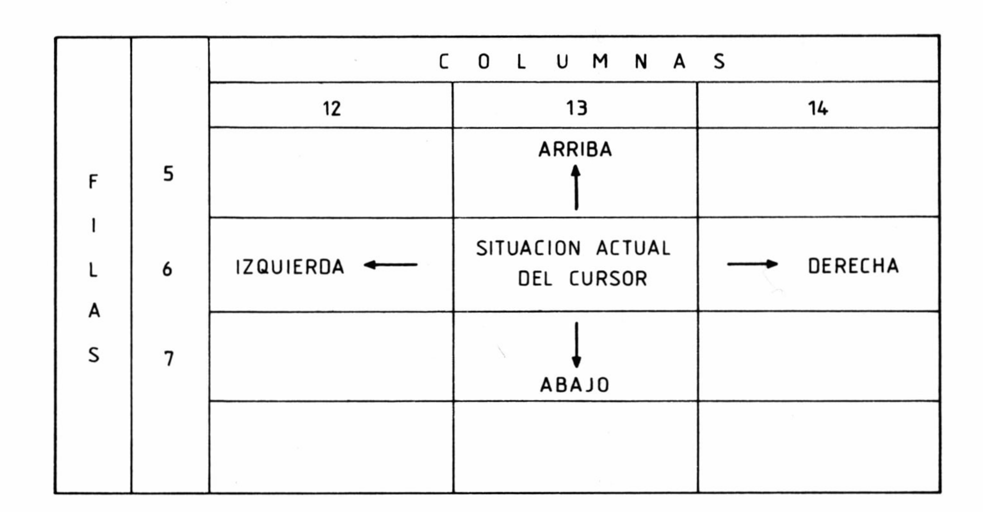


Figura 3 Movimientos básicos del cursor.

— Derecha : Se sitúa en el elemento que está en la misma fila y en una columna más.

En la figura 3 se da un esquema de estos movimientos básicos.

Aparte de estos movimientos básicos son aconsejables los de colocarse en la primera casilla, es decir, la fila 1 y la columna 1. A esta operación la denominaremos inicio de tabla.

Y colocarse en la última casilla. En nuestro supuesto en la columna 15 y la fila 30. A esta operación la denominaremos fin de tabla.

Estos movimientos básicos generan a su vez un movimiento del visor. En efecto, cuando estamos en el borde superior y queremos mover el cursor hacia arriba es necesario que todo el visor se desplace hacia arriba. Lo mismo puede decirse en el caso de estar situados en el borde inferior y deseamos mover el cursor hacia abajo. Cuando estamos situados en el borde izquierdo y deseamos ir más hacia la izquierda. O cuando estamos en el borde derecho y deseamos movernos hacia la derecha.

En todos estos movimientos es necesario tener en cuenta que, cuando estamos en los bordes de la tabla, no se pueden realizar.

Otra condición que permite ganar tiempo en los movimientos es que mientras nos mantengamos en los límites marcados por el visor no es necesario imprimir cada vez todo el visor.

12.4.2 Rutina de visualización del cursor

Una vez probado y grabado el programa para la impresión del visor elimine desde la línea 1000 hasta la 1040, pues son las que se utilizan para comprobar el funcionamiento del programa.

Para llevar el control de dónde está el cursor se deben introducir dos variables que son *f*, para memorizar la fila de la tabla y la *c* para memorizar la columna de la tabla. Estas dos variables nos dan la posición lógica del cursor sobre la tabla.

También se precisa conocer la posición física del cursor sobre la pantalla. Según como se encuentre situado el visor, la posición de la tabla no permite calcular dónde debe situarse el cursor en la pantalla. Se utilizan dos variables *fp* y *cp*, que memorizan la fila y la columna en la pantalla.

La primera operación a realizar es colocar el cursor en algún sitio. Se fija inicialmente en el extremo superior izquierdo de la pantalla. Por lo tanto, las variables *fp* y *cp* deben colocarse al valor 1. En la tabla de valores se hace la misma suposición y también se coloca en la primera fila y la primera columna; por lo tanto se colocan *f* y *c* a 1.

Una vez situado el cursor en estas condiciones hay que mostrar el visor desde la primera fila y columna de la tabla.

En la figura 4 se muestran las modificaciones que hay que hacer en el programa anterior. Recuerde que se debe borrar desde la línea 1000 hasta la línea 1040. Hágalo y vaya siguiendo estas explicaciones consultando en la figura 4. No escriba todavía.

La operación de iniciar el cursor tanto en la parte física como en la lógica y visualizar el visor se realiza en las líneas 1000, 1010 y 1020.

A continuación se procede a visualizar el cursor en la posición inicial y a partir de aquí recibir órdenes desde el teclado para poder realizar los movimientos.

La visualización del cursor consiste en imprimir con el fondo inverso el elemento de la tabla que indican las variables *f* y *c*, en la posición de la pantalla que indican las variables *fp* y *cp*.

Una vez visualizado es necesario esperar que el usuario pulse una tecla para indicar qué operación desea realizar con el elemento que está en el cursor. Esta espera se realiza mediante la combinación de las instrucciones PAUSE y INKEY$.

Una vez recibida la orden es necesario volver a colocar el elemento en forma normal, es decir, con fondo blanco y letras negras; se deja así este elemento igual que antes de que el cursor se sitúe sobre él.

Debido a la complejidad de esta operación se utiliza una subrutina para la visualización del cursor.

La figura 4 muestra esta subrutina desde la línea 9000 hasta la 9040.

La subrutina se llama desde la instrucción 1510, que es la primera instrucción del bucle que recoge las órdenes para realizar el movimiento. Este bucle se inicia en la línea 1500 como indica el comentario.

Antes de seguir con la estructura de las órdenes del programa estudiemos la subrutina de posicionamiento del cursor.

La subrutina se inicia en la línea 9000, como ya se ha dicho, con un comentario para indicar que se trata de una subrutina y qué función realiza.

La línea 9010 es una instrucción PRINT que se inicia con una llamada a la función AT para situar el cursor en el lugar que le corresponde.

En primer lugar hay que colocarlo en la fila de la pantalla *fp*. Debe tener en cuenta que en el ZX-Spectrum a la primera línea de pantalla se le asigna el valor 0; ésta contiene la cabecera de columna. Luego el valor de la variable *fp* nos indica la fila física que requiere la función AT.

El cálculo de la columna es más complicado. De hecho los datos empiezan en la columna siguiente al valor indicado por *abc*, que es el ancho de la cabecera de filas. Por otra parte cada columna tiene un ancho de caracteres *ac*. Si estamos en la primera columna física sólo nos hace falta

```
1000 REM Impresión pantalla inicial
1010 LET f=1 : LET c = 1 : LET fp=1 : LET cp=1
1020 LET q=f : LET p = c : GO SUB 9500

1500 REM Inicio del bucle de ordenes
1510 GO SUB 9000 : REM Control del cursor
1520 IF a$ = "f" OR a$ ="F" THEN GO TO 9999
1530 IF a = 8  THEN GO SUB 8000 : GO TO 1500
1540 IF a = 9  THEN GO SUB 8100 : GO TO 1500
1550 IF a = 11 THEN GO SUB 8200 : GO TO 1500
1560 IF a = 10 THEN GO SUB 8300 : GO TO 1500
1990 BEEP 0.25,10 : GO TO 1500

8000 REM Izquierda
8010 IF c= 1 THEN RETURN
8020 LET c = c -1 : LET cp = cp -1
8030 IF cp > 0 THEN RETURN
8040 LET q= f-fp+1 : LET p=c : GO SUB 9500
8050 LET cp = cp + 1
8060 RETURN
8100 REM Derecha
8110 IF c= tc THEN RETURN
8120 LET c = c +1 : LET cp = cp +1
8130 IF cp <= tpc THEN RETURN
8140 LET q= f-fp+1 : LET p=c-tpc+1 : GO SUB 9500
8150 LET cp = cp - 1
8160 RETURN
8200 REM Arriba
8210 IF f = 1 THEN RETURN
8220 LET f = f -1 : LET fp = fp -1
8230 IF fp>0 THEN RETURN
8240 LET q = f:LET p= c-cp+1:GO SUB 9500
8250 LET fp = fp+1
8260 RETURN
8300 REM Abajo
8310 IF f = tf THEN RETURN
8320 LET f = f +1 : LET fp = fp +1
8330 IF fp<= tpf  THEN RETURN
8340 LET q = f-tpf+1 :LET p= c-cp+1:GO SUB 9500
8350 LET fp = fp-1
8360 RETURN

9000 REM Subrutina para el control del cursor
9010 PRINT AT fp,(cp-1)*ac+acb;"«";t$(f,c);"»";
9020 PAUSE 0 : LET a$=INKEY$ : LET a = CODE a$
9030 PRINT AT fp,(cp-1)*ac+acb;t$(f,c);
9040 RETURN
```

Figura 4 Adiciones y modificaciones al programa para realizar el movimiento del cursor.

sumar *acb*; si estamos en la segunda columna sólo hay que sumar *ac* y *acb*; en la tercera hay que sumar *ac* más dos veces *acb* y en la cuarta *acb* más tres veces *ac*.

La fórmula que realiza este posicionamiento en la columna es

acb+(cp−1)*ac,

en el supuesto que estamos utilizando, en donde, *acb* es 4 y *ac* es 7, la tabla siguiente muestra el comportamiento para los valores de las columnas físicas *cp* que va desde 1 a 4.

cp	cp−1	(cp−1)*ac	(cp−1)*ac+acb
1	0	0	4
2	1	7	11
3	2	14	18
4	3	21	25

Una vez realizada la situación con la función AT, se introduce un texto vacío que sirve para colocar en la pantalla en video inverso (las indicaciones " ‹‹ "y" ›› " se han explicado en la práctica anterior, aquí significan lo mismo), se coloca a continuación el elemento de la tabla señalado por las variables *f* y *c* y se vuelve la escritura a situación normal, es decir, fondo blanco y letras negras.

La línea 9020 recoge la tecla pulsada por el usuario. En primer lugar la instrucción PAUSE 0 para que se detenga el programa hasta que no se pulse una tecla.

A continuación se sitúa en *a$* la tecla pulsada mediante la función INKEY$. Finalmente se calcula el valor de *a* sacando el número en código ASCII de la tecla pulsada. Este último caso es necesario para poder analizar las teclas que no son letras o símbolos, tales como las flechas de movimiento o la tecla de fin de línea.

La instrucción 9030 coloca el elemento que está como cursor en su forma normal de visualización. Se utiliza la función AT de la misma forma que en la instrucción 9010 comentada más arriba y se imprime el valor del elemento de la tabla correspondiente a la fila *f* y la columna *c*.

La línea 9040 devuelve el control al programa principal.

Escriba ahora el programa de la figura 4.

Antes de seguir adelante se prueba esta subrutina del modo siguiente: Primero, se coloca en la línea 1520 una instrucción STOP.

Se ejecuta el programa y el cursor se sitúa en el extremo superior izquierdo de la pantalla indicando el elemento de fila y columna 1.

A continuación toque una tecla, el cursor desaparece y le aparece el mensaje

STOP statement, 1520:1

y la pantalla queda sin cursor.

Realice ahora la siguiente instrucción en modo inmediato

```
LET fp = 20 : LET cp = 4 : GO SUB 9000
```

el cursor se debe situar ahora en el extremo inferior de la pantalla, pero indicando en su interior el elemento lógico correspondiente a la primera fila y columna (1,1).

Toque una tecla para finalizar.

Ejecute las instrucciones siguientes y le debe llevar a los lugares indicados.

```
LET fp = 1 : GO SUB 9000
```

lleva el cursor al extremo superior derecho, no se da cp pues no ha variado respecto a la prueba anterior.

```
LET fp=20 : LET cp=1 : GO SUB 9000
```

lleva el cursor al extremo inferior izquierdo.

```
LET fp = 10 : LET cp = 2 : GO SUB 9000
```

lleva el cursor a la fila 10 y la columna 2.

En todos los casos el elemento que se ha visualizado es el correspondiente a la primera fila y columna, pues no se han modificado las variables *f* y *c*.

Si no se produce este comportamiento repase las instrucciones añadidas.

12.4.3 El bucle de órdenes

En primer lugar borre la instrucción 1520 que contiene un STOP para realizar las pruebas de situación del cursor y cópiela otra vez tal como está en la figura 4.

Esta fase consiste en analizar la tecla pulsada por el usuario para emprender la acción que sea necesaria.

En el caso de este programa las operaciones que deseamos realizar de momento son las operaciones fundamentales de movimiento del cursor. Es decir, se precisa detectar las cuatro órdenes de cursor arriba, abajo, derecha e izquierda.

Siempre hay que tener en cuenta la orden de finalizar que debe planificarse en cualquier programa.

La tabla que se da a continuación define la asociación de teclas a una acción determinada.

f o F	Finalizar el programa.
Flecha arriba	Mover el cursor hacia arriba.

Flecha abajo	Mover el cursor hacia abajo.
Flecha izquierda	Mover el cursor hacia la izquierda.
Flecha derecha	Mover el cursor hacia la derecha.

Las líneas 1520 hasta la 1590 de la figura 4 se destinan a distribuir el control a las operaciones según la orden recibida.

La línea 1520 analiza si la tecla pulsada es la f mayúscula o minúscula. En caso afirmativo se finaliza el programa.

La línea 1530 analiza si la tecla pulsada es la flecha hacia la izquierda. Observe que se pregunta si el código ASCII de la tecla es el 8 que corresponde al de la flecha a la izquierda.

Este número se puede saber mediante la tarjeta del teclado, que está incluida en el primer volumen de esta Enciclopedia, o bien, en el capítulo 7 se han estudiado programas sencillos que permiten averiguar los códigos de estas teclas.

Si la respuesta es afirmativa se envía el control a la subrutina 8000 que se encarga de hacer los cálculos y las acciones oportunas. Cuando se retorna de la subrutina se envía a la línea 1500 para iniciar otra vez el bucle.

Las líneas 1540, 1550 y 1560 hacen una función muy similar, sólo que se envía el control a una subrutina distinta cada vez según la función a realizar.

La línea 1990 se coloca para recoger una tecla que no tiene ningún significado como acción en el entorno de este programa, por ejemplo si se pulsa una Q.

Esta línea utiliza la instrucción BEEP, que aún no se ha estudiado. La función de esta instrucción es producir un sonido para indicar que se ha pulsado una tecla que no se entiende. No se preocupe por el significado de los números, cópiela y más adelante conocerá el significado exacto.

Una vez hemos advertido mediante el sonido que no se entiende la orden, reciclamos el proceso a la línea 1500.

Se preguntará cómo es que se ha utilizado para finalizar el bucle el número 1990. La razón es que dejamos mucho espacio entre la última orden analizada y la advertencia de error, pues a medida que el programa se completa con todas las funciones hay que añadir más órdenes.

Esto debe ser una política general de estos bucles de órdenes, que son sumamente frecuentes en todos los programas un poco complejos.

La estructura de esta parte es muy importante, pues es la manera como los programas recogen órdenes del teclado y bifurcan a las subrutinas que realizan la función deseada.

En este ejemplo, en concreto, el bucle de órdenes consta de las partes siguientes:

- Enviar a la rutina de situación de cursor para establecer la interacción con el usuario.
- Analizar las teclas utilizadas para emprender la acción oportuna. En nuestro caso tenemos cinco órdenes: finalizar, cursor arriba, cursor abajo, cursor a la izquierda y cursor a la derecha. Después de realizar la acción se recicla a la petición de una nueva orden.

— Producir un sonido para indicar error, o mejor, ignorancia de la orden y envío a la petición de una nueva orden.

12.4.4 Cálculo del movimiento del cursor

La última tarea que se debe emprender es realizar el cálculo del movimiento del cursor.

Nos vamos a centrar en primer lugar en el movimiento hacia la izquierda. Los demás tienen razonamientos muy similares. Le advertimos que ponga mucha atención para distinguir cuándo hablamos de las filas o columnas de la tabla y cuándo hablamos de las filas o columnas de la pantalla.

Se parte de que estamos situados en la fila *f* de la tabla y en la columna *c* de la tabla. Por otra parte, el cursor está situado en la fila *fp* de la pantalla y en el columna *cp* en la pantalla.

Según el esquema de la figura 3 ir hacia la izquierda significa dejar las variables *f* y *fp* sin tocar, pues nos mantenemos en la misma fila. En cambio, hay que disminuir en 1 las variables *c* y *cp*, que indican la columna de la tabla y de la pantalla respectivamente.

Esta primera idea debe modificarse un poco, pues hay que tener en cuenta, insistimos, si estamos en los bordes de la tabla o de la pantalla.

El razonamiento es el siguiente:

— Si estamos en el borde izquierdo de la tabla no podemos emprender ningún tipo de acción. Es imposible ir más a la izquierda pues no hay datos.

— En caso contrario disminuimos en uno los valores de *c* y *cp*.

— Si al disminuir *cp* no hemos salido del borde de la pantalla se da por finalizada la acción.

— En caso contrario, nos hemos salido de los bordes de la pantalla; es preciso reimprimir toda la pantalla, ya que se debe correr el visor hacia la izquierda.

La conversión de este razonamiento a lenguaje BASIC está hecha en las líneas 8000 hasta 8060 de la figura 4.

La línea 8010 pregunta si estamos en el borde izquierdo de la tabla, es decir, si *c* es igual a 1. Si la respuesta es afirmativa se procede al retorno al programa principal.

En caso contrario se continúa en la línea 8020. En ella se disminuyen en 1 el valor de *c*, columna de la tabla, y *cp*, columna de la pantalla.

En la línea 8030 se pregunta si *cp* es mayor que cero es equivalente a decir si vale 1 o más. En este caso, nos hemos mantenido en el interior del visor y no hay que hacer ninguna acción más.

En caso contrario, es decir, que *cp* valga cero, es necesario correr el visor hacia la izquierda. Para ello se calculan las variables *q* y *p*, que sirven como argumentos para la subrutina de impresión del visor.

El cálculo de *p* es fácil, pues sabemos que *c* es la columna que debe iniciar la parte izquierda del visor.

El cálculo de la fila es un poco más complicado, ya que debe enviar a

Figura 5 Situación del visor sobre la tabla de datos con el cursor situado en la columna 4 y en la fila 15 de la tabla.



la subrutina la fila que está en primer lugar en la pantalla. El cálculo es, como estoy en *f* de la tabla y *fp* en la pantalla, la primera fila de la pantalla es la fila *f*−*fp*+1 de la tabla.

La línea 8040 calcula estos valores y envía a imprimir la pantalla.

Cuando se retorna, se incrementa el valor de *cp* para corregir la disminución que se ha producido en la línea 8020. En otras palabras, si estamos en la columna 1 de la pantalla, al retroceder el visor hacia la izquierda, el cursor se queda en la columna izquierda.

Para acabar de comprender vamos a representar gráficamente el caso de que tuviéramos situado el cursor en la columna 4 y en la fila 15 de la tabla y en la columna 1 y en la fila 5 del visor que nos representa la pantalla. Gráficamente la situación sería la que puede ver en la figura 5. En ella puede observar la tabla completa con sus 15 columnas y con sus 30 filas. Sobre ella hemos colocado la pantalla que nos hace de visor y que nos permite inspeccionar la parte correspondiente de la tabla. El visor está constituido por la parte de tramado claro, y el tramado oscuro pertenece a lo que sería el diseño de la pantalla. Si observa ahora detenidamente la figura 5 puede ver claramente el supuesto de que partíamos. Es decir, que el cursor se encuentra en la columna 4 y en la fila 15 de la tabla y en la primera columna (la situada más a la izquierda) y en la fila cinco de la pantalla.

Supongamos ahora que deseamos inspeccionar lo que hay en la fila 15 y en la columna 3 de la tabla. Para ello, basta que el cursor se corra una columna hacia la izquierda en el visor. Sin embargo, esto no es posible, puesto que el cursor está en la columna 1 del visor y correrlo hacia la izquierda significaría pasarlo a la columna cero, que no existe. Por tanto, lo que debemos hacer es correr todo el visor hacia la izquierda una columna. Con esto, que lo hubiera sido la columna cero del visor, será nuevamente la columna 1, pero situada sobre la columna 3 de la tabla. Es lo que puede ver en la figura 6. Observe que el *visor* (tramado claro) comienza en la columna 3 y en la fila 15−5+1 de la tabla; es decir, en la fila 11.

Una vez comprendido este movimiento es fácil comprender los demás. De la línea 8100 hasta la 8360 están las instrucciones BASIC del resto de movimientos.

Veamos el movimiento hacia la derecha. En este caso, estar al borde de la tabla significa que *c* es igual a *tc*, pregunta que se realiza en la línea 8110.

A continuación se procede a aumentar en uno la columna de la tabla y de la pantalla. El movimiento es inverso al anterior. Antes se restaba, ahora se suma.

Se pregunta si la columna de pantalla es menor o igual a *tpc*, que es el número de columnas máxima que acepta la pantalla. Si la respuesta es afirmativa se acaba la operación.

En caso contrario, hay que mover el visor hacia la derecha. El cálculo de la fila de la tabla con que se inicia la pantalla es el mismo que en el caso anterior del movimiento hacia la izquierda. Ha cambiado el caso de la columna, sabemos que como hay que correr el visor, la columna *c* debe aparecer en el borde derecho de la pantalla. Si la pantalla tiene *tpc* columnas, la que está en el borde izquierdo es *c*−*tpc*+1.

Finalmente se reajusta el valor de la columna de pantalla al borde izquierdo.

Figura 6 Retroceso del cursor una columna hacia la izquierda.

Figura 5

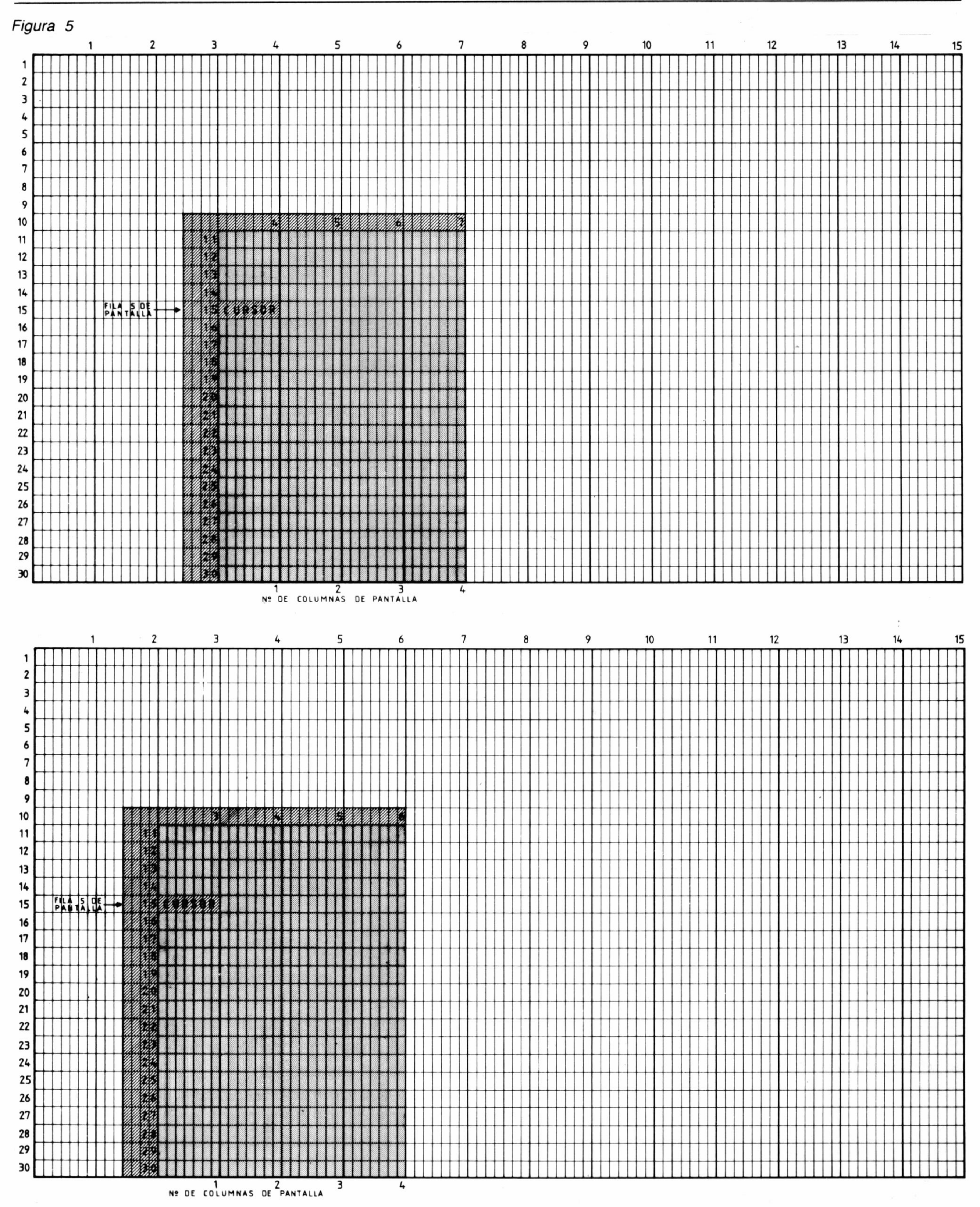


Figura 6

La subrutina de realizar la operación de subir hacia arriba tiene un parecido muy notable con la de ir hacia la izquierda. En ésta se hace sobre columnas; en la de arriba se hace sobre filas.

Esta misma similitud se observa entre la rutina de ir hacia la derecha y la rutina de ir hacia abajo, sólo que se intercambian los papeles de filas y columnas.

12.4.5 Las operaciones de inicio y final de tabla

Una vez tecleado todo el programa, se ejecuta. Las flechas deben proprocionarnos el movimiento del cursor de acuerdo con nuestros deseos.

Debemos hacer un text exhaustivo de movimientos y observar que podemos subir y bajar no sólo el cursor, sino también que el visor se mueve adecuadamente o no cuando está en los bordes de la pantalla y tabla respectivamente.

Si el funcionamiento no es correcto compruebe sobre todo las rutinas del movimiento del cursor. Es fácil cometer errores de copia debido a la importancia que tiene cada uno de los símbolos. La sencillez aparente no significa que sea fácil.

No olvide de introducir alguna orden incorrecta para oír el sonido de error ni de acabar con la tecla f tal como está planificado. El sonido también lo oirá si se empeña en mover el cursor cuando se está reescribiendo la pantalla.

De hecho nos habíamos propuesto dos operaciones más que es situarse al inicio de la tabla y al final de la tabla.

Aparte de las operaciones en sí mismas, es importante que se de cuenta de lo relativamente fácil que es añadir estas nuevas operaciones sobre el programa que tenemos hecho.

En primer lugar hay que definir qué teclas nos proporcionan la acción. Se elige la p (de primero) para ir al principio de la tabla. Se elige la u (de último) para ir al final de la tabla.

Luego hay que añadir en el bucle de órdenes la bifurcación adecuada para tratar estos dos casos. Las instrucciones son las siguientes:

```
1570 IF a$="p" OR a$="P" THEN GO SUB 8400 : GO TO 1500
1580 IF a$="u" OR a$="U" THEN GO SUB 8500 : GO TO 1500
```

Observe qué interesante ha sido colocar el 1990 como número de línea en el diseño del bucle de órdenes.

También observe que en estos casos es mejor trabajar con a$ que con el código ASCII; la estructura del programa nos lo permite. Se aceptan las letras tanto en mayúsculas como en minúsculas.

Hay que diseñar ahora las dos operaciones. Estas operaciones son sencillas, ya que colocarse a inicio quiere decir que *f, c, fp* y *cp* deben colocarse a 1. Por otra parte, para evitar complicaciones imprimiremos de nuevo el visor.

Las instrucciones son las siguientes:

```
8400 REM Inicio de la tabla
8410 LET f=1 : LET c=1 : LET fp=1 : LET cp=1
8420 LET q=f : LET p=c : GO SUB 9500
8430 RETURN
```

Compare estas instrucciones con las de números 1000 hasta la 1020 y vea que son idénticas. De hecho se inicia el programa llevando la tabla al inicio. Por eso, se pueden sustituir las instrucciones 1010 y 1020 por una llamada a 8400.

La operación de llevar el cursor al final de la tabla es muy semejante. Se colocan *f, c, fp* y *cp* a sus valores máximos, es decir, *tf, tp, tpf* y *tpc*. Además hay que imprimir el visor. La única diferencia es el cálculo de dónde debe empezar el visor.

El cursor se sitúa en el extremo inferior derecho de la pantalla y tenemos que calcular qué fila y columna de la tabla corresponde al extremo superior izquierdo. Basta restar a la columna y fila de la tabla el valor de la columna y fila física añadiendo 1.

Las instrucciones son:

```
8500 REM Fin de tabla.
8510 LET f = tf : LET c = tc : LET fp = tpf : LET cp = tpc
8520 LET q = f-fp+1 : LET p = c-cp+1 : GO SUB 9500
8530 RETURN
```

Después de añadidas estas instrucciones compruebe el funcionamiento del programa con estas nuevas operaciones.

Es el momento además de guardar el programa en cassette con el nombre «p2».

12.5 PRACTICA 3. ENTRADA DE DATOS

Con la práctica anterior hemos conseguido poder desplazarnos por la tabla con gran comodidad mediante la utilización de los conceptos de cursor y visor.

El objetivo ahora es modificar el contenido de la tabla.

La tabla tiene los elementos con un contenido provisional para poder observar con detalle los movimientos del visor.

Este objetivo modifica dos aspectos del programa:

— En primer lugar es necesario eliminar la parte de relleno provisional, es decir, las líneas 500 y sucesivas hasta la 580. Con estas modificaciones la tabla queda llena de blancos o de cadenas vacías. Este aspecto es muy fácil de conseguir porque el relleno de la tabla se hace por motivos de prueba del programa.

— En segundo lugar hay que añadir una nueva operación para indicar al sistema que se quieren entrar datos. Hay que modificar el bucle de órdenes y construir una subrutina que haga efectivamente la entrada.

Para abordar el segundo aspecto, en primer lugar definiremos la tecla que nos da acceso a la operación de entrada y modificación de una casilla. Se elige la tecla EDIT, que tiene la misma función que en un programa: variar o modificar el contenido de un elemento de la tabla. Por tanto, cuando tengamos que entrar datos, después de coloca el cursor en el lugar que deseemos, debemos pulsar la tecla EDIT.

Se introduce entonces en el bucle de órdenes con la instrucción siguiente:

```
1600 IF a = 7 THEN GO SUB 7000 : GO TO 1500
```

Observe que se ha numerado con 1600 para dejar espacio y tener una separación clara entre un tipo de operaciones y otros.

Esta operación genera la necesidad de construir la subrutina 7000, que es la entrada de datos propiamente dicha.

Esta subrutina puede ser muy simple. En una primera aproximación nos basta con un INPUT.

Las instrucciones son las siguientes:

```
7000 REM Entrada de datos
7010 INPUT t$(f,c)
7020 RETURN
```

Este mecanismo tiene el inconveniente de que hay que entrar de nuevo el contenido de la celda ya que no es posible modificar el ya existente como ocurre con la tecla EDIT en los programas.

La resolución de este problema no es sencilla, de hecho hay que iniciar el diseño de un editor de líneas con una complicación parecida al de la tabla, ciertamente algo menor. De momento dejaremos esta posibilidad para otra ocasión.

Una solución intermedia es colocar el valor de la celda al lado del valor a entrar de donde se modifica la instrucción INPUT del modo siguiente:

```
7010 INPUT "Valor "+t$(f,c)+ " a:";t$(f,c)
```

de esta manera el valor que está en este momento aparece junto al INPUT, lo que da más facilidad para el proceso de entrar.

Un aspecto que hemos descuidado es que en la tabla deben aparecer las cadenas ajustadas a la derecha, por lo tanto después de entrar el valor es necesario utilizar la subrutina de ajustar a la derecha.

Las instrucciones son:

```
7000 REM Entrada de datos
7010 INPUT "Valor "+t$(f,c)+ " a:";a$
```

```
7020 GO SUB 9900 : LET t$(f,c) = a$
7030 RETURN
```

Sin embargo, esta estructura nos permite introducir cualquier cadena de caracteres en el interior de la tabla. Ejecute el programa y pruebe de entrar cualquier texto.

Esta entrada no cumple especificaciones, ya que nos han dicho que la tabla debe ser de números enteros y positivos.

Hay que incluir en la subrutina esta comprobación. Este requisito es equivalente a que la cadena sólo contenga números.

Esta condición es fácil de colocar, pero el tener que comprobar nos introduce una complicación adicional, que es que no se puede hacer el RETURN si la entrada no es correcta. Por otra parte, debemos diseñar algún mecanismo para poder salir y dejar las cosas como estaban, si en algún momento por equivocación damos la tecla EDIT y no queremos hacer nada. Debemos, pues, pensar en una tecla que nos permita salir de la entrada sin realizar ninguna modificación.

Finalmente, la tabla está rellena de blancos inicialmente, un valor de la tabla sin ningún número debe ser correcta también.

La figura 7 contiene las instrucciones para realizar esta operación. Puede introducirla, teniendo en cuenta los comentarios que siguen.

La línea 7010 entra el valor en la variable a$ recordando el valor que tiene en este momento el elemento de la tabla.

```
Eliminar líneas 500, 510, 520, 530, 540,
                550, 560, 570 y 580.


1600 IF a=7 THEN GO SUB 7000 : GO TO 1500


7000 REM Entrada de datos
7010 INPUT "Valor "+t$(f,c)+" a:";a$
7020 PRINT AT 21,1;"                              ";
7030 IF a$ = " STOP " THEN RETURN
7040 LET l = LEN (a$)
7050 FOR i = 1 TO l
7060 LET b$ = a$(i TO i)
7070 IF "0"<=b$ AND b$<="9" THEN GO TO 7100
7080 PRINT AT 21,1;"Cifra incorrecta";
7090 GO TO 7000
7100 NEXT i
7110 IF l <= ac THEN GO TO 7140
7120 PRINT AT 21,1;"Numero demasiado grande";
7130 GO TO 7000
7140 GO SUB 9900 : LET t$(f,c)=a$
7150 RETURN
```

Figura 7 Modificaciones para la entrada de datos.

La línea 7020 escribe blancos en la línea 21 de pantalla por si ha quedado algún error escrito. Cuando más tarde se entran elementos incorrectos se informa con un error en esta línea.

La línea 7030 sale de esta operación si la tecla pulsada es el STOP. Recuerde que hay que pulsar la tecla situada sobre la A con SYMBOL SHIFT apretado. No sirve escribir todas las letras de STOP.

Esta línea añade la especificación de salir de la operación si no queremos hacer nada.

La línea 7040 busca la longitud del texto entrado.

La línea 7050 inicia un bucle para repasar cada uno de los símbolos entrados en a$.

La línea 7060 selecciona en b$ la letra correspondiente a la posición i de la cadena de a$ mediante el operador de fragmentación. Se hace así para evitar calcularlo dos veces en la instrucción siguiente.

La línea 7070 comprueba si b$ es una cifra numérica. En caso afirmativo envía el programa a la línea 7100, que es la continuación del bucle para inspeccionar el carácter siguiente.

Si la respuesta es negativa sigue en la línea 7080, que imprime un mensaje de cifra incorrecta y en la línea 7090 se recicla el programa a una nueva entrada (línea 7000).

En la línea 7110 se pregunta si la longitud es menor o igual que el ancho de campo. Si la respuesta es negativa sigue por la línea 7120 colocando un error de que el número es demasiado grande y enviando el control a la línea 7000 para hace una nueva entrada.

Esta precaución se toma no por razones de visualización. Recuerde que al encho de caracteres del elemento está fijado y si se entran más caracteres se cortan oportunamente. La razón es que el operador va a perder cifras al colocarlo en la tabla y se le advierte de esta circunstancia.

Si la longitud es la correcta se envía a la línea 7140, en donde se ajusta el número a la derecha mediante la subrutina 9900 y se coloca a$ en la tabla.

Finalmente, la línea 7150 devuelve el control al programa principal.

Observe que en el caso de entrar la cadena vacía el resultado es equivalente a borrar el elemento. En efecto, no pasa por el bucle porque el valor final es más pequeño que el inicial. Por otra parte cumple la condición de la línea 7110 y alcanza el final de la subrutina.

Una vez tecleadas estas subrutinas ejecute el programa.

En primer lugar la tabla aparece toda en blanco. Apriete la tecla EDIT y se inicia la subrutina 7000.

Conteste a la pregunta mediante unas letras o un número con decimales y el programa debe responderle con el error cifra incorrecta.

Teclee ahora un número de más de 7 cifras y observe el error que le aparece.

Finalmente, teclee un número correcto y se finaliza la subrutina correctamente.

Mueva el cursor y vuelva sobre el número anterior.

Le advertimos que si aprieta dos veces seguidas la tecla EDIT se le borrarán las comillas donde se entra el dato. Puede escribirlas de nuevo y colocar el cursor entre ellos.

Apriete la tecla EDIT. En la pregunta que acompaña el INPUT le aparece el valor que tiene el elemento.

Ahora apriete el STOP (SYMBOL SHIFT + tecla A) y vuelve a la tabla sin modificar el número.

Una vez realizadas las pruebas y confirmado el correcto funcionamiento del programa guárdelo en un cassette con el nombre «p3».

12.6 PRACTICA 4. GUARDAR DATOS EN CASSETTE

Hemos conseguido una manera cómoda de entrar datos en una tabla, modificarlos y visualizarlos. Este trabajo se puede aprovechar en otros programas para que se realicen cálculos.

Tenemos dos alternativas, o bien incluir el programa de cálculo en este programa o bien guardar los datos en un medio magnético y el programa, que calcula y lee los datos de este medio magnético.

La primera alternativa es válida, pero suele ser complicado añadir una fase de cálculo en un programa complejo como el que disponemos en este momento.

La segunda alternativa es más atractiva porque el programa de cálculo será más limpio y, por lo tanto, más comprensible. Además, a una misma tabla se le pueden realizar cálculos distintos con programas distintos y el mismo programa nos permite trabajar con tablas distintas.

El único medio magnético de que disponemos es la cinta de cassette y lo utilizaremos para grabar esta tabla.

En primer lugar el ZX-Spectrum dispone de dos instrucciones básicas (LOAD y SAVE) para realizar esta transferencia de datos al cassette.

La manera como se utilizan para guardar son las siguientes:

```
LOAD "numeros" DATA n()
LOAD "textos"  DATA n$()
SAVE "numeros" DATA n()
SAVE "textos"  DATA n$()
```

el significado de LOAD y SABE ya lo conocemos por el capítulo 8 del volumen II de la Enciclopedia. El primero significa cargar desde cassette a memoria y el segundo guardar desde memoria a cassette.

En todos los casos le sigue un nombre o una variable textual que indica que se desea cargar o grabar una porción de cinta magnética con este nombre.

En las instrucciones hemos distinguido estos nombres para indicar que en unos casos la transferencia es de números y en otros de texto.

Las instrucciones van seguidas de la palabra DATA para indicar al BASIC que se quieren transferir datos.

Finalmente, aparece el nombre de la tabla o lista que se quiere transferir. Hemos escogido el nombre de *n()* o *n$()* para una tabla de números o una de textos respectivamente. Cuando se utiliza una tabla de números se transfieren números y cuando se utiliza *n$()* se transfiere texto. En todo caso, en la práctica que estamos realizando se ha partido del supuesto de que en la tabla entramos números enteros.

El efecto de estas instrucciones es igual a los del LOAD y SAVE de programas y la nomenclatura de los nombres es también la misma. La única diferencia es que en los primeros se guardan las instrucciones de los programas y en la segunda se guardan los datos contenidos en una tabla.

En nuestro programa lo único que hace falta es añadir las operaciones que nos manipulan el cassette.

En una primera fase definiremos qué teclas desencadenan estas operaciones.

Tomamos para la operación de carga la letra j tanto en mayúsculas como en minúsculas. Se hace esta elección porque corresponde a la tecla de la palabra clave LOAD.

Para la operación de guardar se utiliza la letra s (S), porque la palabra clave SAVE se encuentra sobre esta tecla.

Finalmente, aunque no está relacionada directamente con el cassette, es necesaria una operación que nos coloque la tabla en blanco totalmente. Esta operación es necesaria por la razón siguiente: Editamos una tabla de unos datos determinados. Guardamos el contenido en el cassette. Deseamos ahora iniciar una nueva tabla; la operación de limpiar la tabla sirve para cumplir esta misión. Su función es equivalente al NEW cuando estamos preparando programas. Por ello se toma la letra a (A) como desencadenador de la operación, pues la tecla contiene la palabra clave NEW.

La figura 8 indica las líneas que hay que añadir para tener en cuenta estas operaciones.

Las líneas 1700, 1710 y 1720 se deben colocar dentro del bucle de órdenes para tener en cuenta estas operaciones que llaman a las subrutinas 6000, 6100 y 6200.

La fase siguiente es realizar las subrutinas para esta misión.

```
1700 IF a$="j" OR a$="J" THEN GO SUB 6000 :GO TO 1500
1710 IF a$="s" OR a$="S" THEN GO SUB 6100 :GO TO 1500
1720 IF a$="a" OR a$="A" THEN GO SUB 6200 :GO TO 1500


6000 REM Carga de tabla
6010 INPUT "Cargar la tabla de nombre:";a$
6020 LOAD a$ DATA t$()
6030 GO SUB 9500
6040 RETURN

6100 REM Guardar tabla
6110 INPUT "Guardar tabla de nombre:";a$
6120 SAVE a$ DATA t$()
6130 RETURN

6200 REM Limpiar la tabla
6210 INPUT "Confirma su decisión(S)";a$
6220 IF a$<>"S" AND a$<>"s" THEN RETURN
6230 FOR i = 1 TO tf
6240 FOR j = 1 TO tc
6250 LET t$(i,j) = ""
6260 NEXT j
6270 NEXT i
6280 GO SUB 8400
6290 RETURN
```

Figura 8 Adiciones para cargar y guardar la tabla en cassette.

Comencemos con la subrutina 6000, que pretende cargar una tabla de datos desde el cassette en la memoria del ordenador.

La línea 6010 nos pregunta el nombre de los datos que deseamos guardar. Se almacena en la variable *a$*.

La línea 6020 es la instrucción que carga desde el cassette los datos que tienen el nombre a$ y los coloca, ya que hemos utilizado la palabra DATA detrás, en la tabla *t$*.

Cuando se ejecuta esta instrucción la pantalla queda en la posición de LOAD normal como en un programa. Apriete entonces la tecla PLAY de la grabadora.

Como en el cassette puede existir más de un conjunto de datos aparecen en pantalla los programas y datos, que contiene el cassette. A medida que van pasando va cualificando el tipo de elemento, siendo así cuando se obtiene el mensaje

Program: Es que pasa un programa.

Bytes: Es que pasan números.

Character array: Es que pasan tablas de caracteres.

Estos mensajes aparecen en la pantalla hasta encontrar el que se ha pedido. Una vez se ha encontrado, la pantalla no contiene el visor sino el visor más los caracteres correspondientes a la lista de datos que han pasado por el cassette hasta alcanzar los que hemos pedido.

Por lo tanto, después de hacer una carga es necesario volver a visualizar el visor. La línea 6030 envía a la impresión del visor. Observe que no se calcula ni *p* ni *q*, que son los argumentos que requiere la subrutina del visor. Se mantienen los pedidos en la última vez que se ha pedido la visualización del visor.

Finalmente la línea 6040 retorna al programa principal.

La operación de guardar unos datos en cinta es muy semejante a la anterior.

La ejecución de la línea 6120 desencadena la aparición del mensaje

Start Tape. Then press any key.

que nos informa de que pongamos en marcha la cinta y pulsemos cualquier tecla. Recuerde que debe desconectar el cable que va a EAR y pulsar las teclas REC y PLAY de la grabadora para iniciar la grabación.

Aparecen las líneas típicas de grabación en los bordes de la pantalla hasta que finaliza y vuelve el cursor a aparecer en pantalla. En este momento debe parar la gravadora.

Finalmente, desde las líneas 6200 hasta la 6280 es la operación de limpiar la tabla colocando todos sus elementos a blanco.

En la línea 6210 se pregunta confirmación para esta operación, ya que si se ha pulsado por error podría destruir mucho trabajo realizado. Si se pulsa una s tanto en mayúscula como en minúscula se procede a efectuar la limpieza, en caso contrario se devuelve el control al bucle de órdenes. Esta pregunta se hace en la línea 6220.

En la línea 6230 se inicia un bucle para las filas y en la línea 6240 un bucle para las columnas. En la línea 6250 se llena la matriz de elementos

vacíos, que no es necesario ajustar porque el comportamiento del ZX-Spectrum para rellenar una tabla es añadir blancos, que en este caso nos da igual que sea por la derecha que por la izquierda.

Las líneas 6260 y 6270 cierran el bucle.

La línea 6280 ejecuta la subrutina de ir al inicio de la tabla y la línea 6290 devuelve el control al programa principal.

Pase ahora a la fase de pruebas.

Antes de poder cargar un programa es necesario tener preparados algunos datos en el cassette. Para ello entre algunos valores en la tabla. Una sugerencia es entrar la suma de los índices de los elementos, así en la posición fila 2 y columna 3 se entra un 5 que es la suma de 2 más 3. No es necesario que entre estos números en toda la matriz.

A continuación apriete la tecla S para realizar la operación de guardar la tabla. La operación le pregunta el nombre. Conteste, por ejemplo, *n1*. Le aparece entonces el mensaje de que ponga la grabadora en marcha y apriete cualquier tecla. Hágalo, no se olvide de desconectar el EAR.

Aparecen en la pantalla las típicas rayas que aparecen cuando se graba un programa. A continuación el cursor vuelve a aparecer sobre la tabla.

Pulse la tecla A para borrar toda la tabla. Tenga paciencia, es una operación un poco larga, el ordenador se toma de 45 segundos a un minuto aproximadamente.

Empiece a rellenar la tabla otra vez, pero ahora utilice en cada casilla el número de fila a que pertenece.

Una vez rellenada una parte ejecute otra vez la operación de guardar pero con el nombre *n2*.

En este punto conecte el EAR y apriete la operación de carga, con la tecla J. No es necesario poner a blancos otra vez la tabla, al cargarse ya se escribe encima de lo que tenemos ahora.

Si le pregunta el nombre escriba *n1*.

La pantalla se coloca en el parpadeo típico de carga de un programa. Rebobine la cinta y apriete el PLAY.

La cinta va pasando. Si se se ha rebobinado al principio, aparecen los mensajes siguientes:

Program p1

Program p2

Program p3

Program p4

Character Array n1

y se procede a la carga de los datos.

Un vez finalizado detenga la cinta y observe cómo la tabla está rellena con los datos colocados en la primera tabla editada.

Realice otra vez la operación pero dando como nombre *n2*. No hace falta que rebobine la cinta.

Después de la operación tendrá en la tabla lo que ha editado en último lugar.

Índice

PARTE II. PRACTICAS CON EL ORDENADOR

Capítulo 9

Capítulo 10

Capítulo 11

Capítulo 12



SPECTRUM

3

- La estructura de repetición
- Las instrucciones FOR y NEXT
- Casos de bucles FOR
- Bucle escalonado
- Bucles decrecientes
- Bucles anidados
- Bucles de espera
- GOTO y los bucles
- Posibilidades de los bucles FOR/NEXT
- Listas y tablas
- Homogeneidad
- Nomenclatura
- Reserva de memoria
- La instrucción DIM
- Variables textuales
- Conjuntos dimensionados
- Subíndices
- Elemento cero
- Las instrucciones DATA, READ y RESTORE
- Funciones definidas por el usuario
- Subrutinas
- Modularización de programas
- Biblioteca de subrutinas y aplicaciones
- Números aleatorios
- Un editor de tabla

ceac